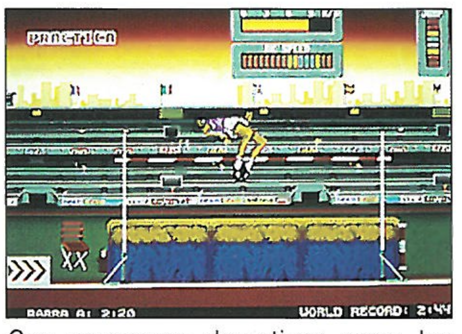

SÓLO PROGRAMADORES
Revista especializada para usuarios de PC
AÑO 2 Nº 6
1250 PTAS.
ESCRITA por desarrolladores para desarrolladores
Entrevista a los programadores de Bit Manager
DE REGALO
El S. Operativo LINUX 2.1.0 + utilidades
Especial sobre los puertos del PC
Y además:
• Curso de 3D.
• Redes Locales
• Curso de UNIX
• Normas de estilo
• Formato MOD
• Introducción V. Basic
Instalación de Slackware LINUX 2.1.0
00006
ORNEO DE PROGRAMACION
000 PTAS. EN PREMIOS!
TOWER
COMMUNICATIONS, S.R.L.

LOS MEJORES PROGRAMAS PARA TU PC



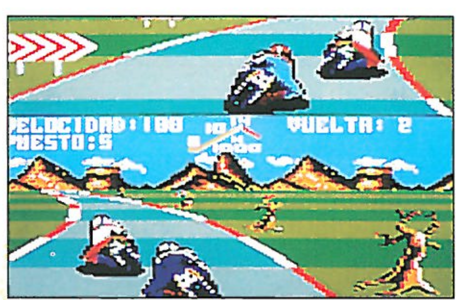
El baloncesto en su máxima expresión, mates, pases, asistencias, etc.



Con programas deportivos como las Olimpiadas, con todas las pruebas.



PACK OPERA 25 contiene el famoso juego de mesa para adultos "La Colmena".



Acción sin límite para toda la familia: carreras de motos, uno o dos jugadores.

PC CD-ROM

PACK OPERA 25

NUEVO

Incluye Libro de Trucos de juegos para PC

TOWER COMMUNICATIONS

DDM DIGITAL DREAMS MULTIMEDIA

25 juegos COMPLETOS en CD-ROM por sólo 2.995 ptas. :

LA ABADIA DEL CRIMEN	LA COLMENA	LA PULGA
BASKET	LIVINGSTONE, SUPONGO	RESCATE EN EL GOLFO
BOXEO	LIVINGSTONE, SUPONGO II	SIRWOOD
CORSARIOS	THE LAST MISSION	SOL NEGRO
FUTBOL	MITHOS	SOLO
GONZZALEZZ	MOT	SOVIET
GUILLERMO TELL	MUTAN ZONE	TRIGGER
GOODY	OLIMPIC GAMES	ULISES
JAI ALAI	POLE 500	

TOWER COMMUNICATIONS S.R.L.

CON LA GARANTÍA DE

DDM DIGITAL DREAMS MU...

Solicita PACK OPERA 25 enviando este cupón o llamando al teléfono (91) 741.26.62 de 9...

Deseo que me envíen PACK OPERA 25 al precio de 2995 ptas + 200 ptas. de gastos de envío.

Nombre y apellidos..Domicilio..Poblaci...

Provincia..C.P ....Fecha de nacimiento..............................Profesión..........

FORMA DE PAGO:

☐ Talón a TOWER COMMUNICATIONS S.R.L. ☐ Contra reembolso (+ 200 Ptas. adicionales de...

☐ Giro Postal nº..de fecha..........................

Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

AMERICAN EXPRESS nº ☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐☐

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto..................





CARTA DEL DIRECTOR

COMUNICACIONES PARA TODOS

Seguramente hace unos años nadie había oído hablar de Internet (calma, el editorial de este mes no está dedicado a este tema, por ahora..), pero el hecho es que desde hace unos meses, es el centro de las miradas de atención de casi todas las revistas de informática. El motivo es muy sencillo, la información a la que los *navegantes* pueden acceder es inmensa y, casi siempre, de última hora. De hecho, corre por ahí la broma de que ahora el mejor programador no es el que realiza las rutinas más rápido, sino el que antes las encuentra ya hechas en Internet. Bueno, el caso es que, como ya anticipaba, no hablaré *por ahora* de Internet.

El mundo de las comunicaciones está sufriendo una verdadera revolución día tras día y lo mejor es que los nuevos avances pueden cambiar las formas y métodos tradicionales de producción. El *teletrabajo* y la *telepresencia* serán posibles en breve gracias a la instalación de líneas de alta velocidad RDSI, que Telefónica tiende estos días sin descanso. Seguramente, dentro de un año, el editorial estará dedicado al monopolio sobre las líneas de comunicaciones y la falta de alternativas. El hecho es que hoy en día sólo una gran empresa, como el estado, puede asumir el coste de recorrer todo el país tendiendo el cable que tardará años en ser amortizado. Y lo malo, es que Telefónica, pese a ser una empresa participada por todos los que pagamos impuestos, funciona como eso, una empresa, y año tras año necesita producir beneficios que salen del bolsillo de los mismos que la financian. Curiosa paradoja ¿no? .

El mes pasado hubo un pequeño conato de rebelión por parte de los centros servidores de Ibertex al experimentar *en sus carnes* una subida desmesurada de la cuota de mantenimiento en las líneas X-25. Pero las causas justas no siempre vencen y tratar de luchar contra Telefónica es como darse golpes contra la pared. Lo mejor es darse la vuelta. Si uno se queja de algo la respuesta es: *te cortamos el teléfono* o *tómalo o déjalo*. Se han olvidado de dónde ha salido el dinero para crear esa macroempresa. Tal vez la solución sea que otra compañía de dimensiones similares, como British Telecom, entrara en nuestro país ofreciendo el mismo servicio para establecer una competencia de la que los usuarios saliésemos beneficiados. La realidad es que a día de hoy, no hay otra cosa para elegir y, tal y como están las cosas, es difícil que cambien al menos durante bastante tiempo. Lástima, porque a mí eso del módem me gustaba desde pequeñito.

MARIO DE LUIS



FEBRERO 1995. Número 6
Es una publicación de



COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

••••••

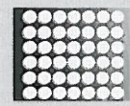
Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Miguel Angel Alcalde

Colaboradores
María Gil, Luis Crespo, Sergio Ríos, Sonia Sancibrián, Luis F. Fernández, Carlos Arias, Cristina de la Fuente, Bernardo García, Agustín Guillén, Fernando J. Echevarrieta, Emilio Postigo, Ignacio Cea, César Astudillo, Fernando de la Villa.

Edición técnica
Sergio Cabrera
Maquetación
Fernando García Santamaría
Asesor Técnico de Autoedición
Antonio López Areosa
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Consuelo Jiménez
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

Depósito legal: M-17806-1994
ISSN: 1134-4792

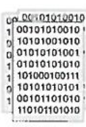
SUMARIO



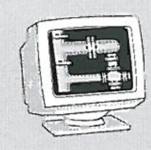















SUMARIO

1er TORNEO DE PROGRAMACION

4 en Raya

En el pasado mes de diciembre, Sólo Programadores *convocó el Primer Torneo de Programación. Debido al gran interés que éste ha suscitado entre la comunidad de programadores, se amplía el plazo de entrega hasta el 15 de marzo.*

El campo de batalla virtual en el que combatirán los programas enviados por los participantes será el tablero del juego Las 4 en Raya. Las aplicaciones que acudan al torneo se distribuirán en cinco grupos, en los que se enfrentarán de dos en dos hasta que sólo quede una "viva" en cada grupo. Cada uno de los cinco supervivientes recibirá como premio un compilador Borland C++ 4.02 en castellano.

Los cinco finalistas se disputarán el honor de ser el mejor en una liguilla clasificatoria en la que lucharán todos contra todos. El vencedor de dicha liguilla obtendrá un diploma en el que se certificará que ha conseguido el primer puesto en el Primer Torneo de Programación de Sólo Programadores.

COMO SE JUEGA A LAS 4 EN RAYA

Este popular juego de mesa de dos jugadores está compuesto por un tablero vertical que posee 7 columnas abiertas en la parte superior, en las que caben 6 fichas por columna. Se forma así una matriz de 6 celdas de alto por 7 de ancho. Hay 42 fichas, 21 blancas y 21 negras. Los participantes van introduciendo sus fichas de una en una de forma alternativa por la parte superior, y éstas caen hasta encontrar fondo o apoyarse en otra. Cuando se forma un línea horizontal, vertical o diagonal de 4 fichas consecutivas del mismo color, el jugador al que pertenezcan esas fichas gana la partida.

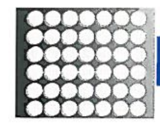
DESARROLLO DE UNA PARTIDA

Un programa árbitro ejecutará al azar uno de los dos programas en liza. Este deberá comprobar que es el primero y utilizará el resto de la partida el 1 como ficha. Al terminar su movimiento acabará la ejecución y volverá al sistema operativo, pasando el turno al otro participante, que averiguará si es el primero o no. En este último caso su ficha será un 2. Se permitirá que creen un sólo fichero cada uno para almacenar la información que consideren necesaria, como por ejemplo la ficha que deben emplear, los tableros, las tácticas, etc. Al terminar cada movimiento el árbitro comprobará si se han conseguido 4 en raya. Si el árbitro detecta cualquier acto erróneo o ilegal, expulsará inmediatamente del torneo al participante.

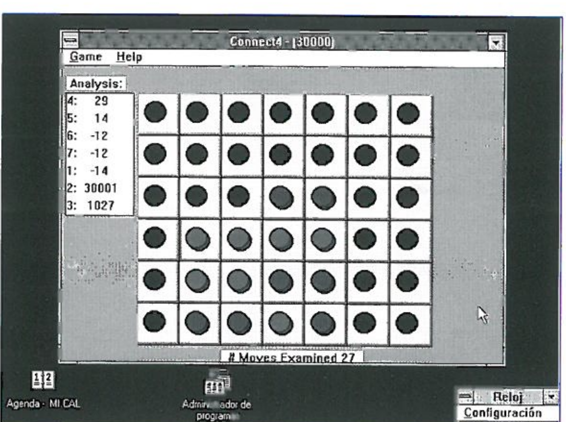
Se jugarán dos partidas en cada encuentro, en las que empezará una vez cada uno. En caso de que haya empate ganará quién menos movimientos haya realizado para vencer a su contrincante. Si sigue habiendo empate se disputará la tercera y definitiva partida.

UN EJEMPLO

En el disquete o en el CD-ROM se encuentra un directorio denominado **TORNEO** en el que hay un pequeño ejemplo de cómo debe ser el programa para aquellos que no sepan cómo empezar. Este programa escribe en pantalla algunos mensajes para ser más didáctico, pero el programa que compita no puede utilizar la pantalla para imprimir ni esperar pulsaciones de teclas. Así mismo se puede ver cómo es el fichero **TABLERO.DAT**. Hay un directorio llamado **DHRYSTON** que contiene varios ficheros. Uno de ellos es un .TXT que dice para qué sirven los otros.



BASES (Aviso: hay una modificación de última hora)



Existen juegos de 4 en Raya de mesa o para ordenador, como éste que funciona bajo Windows.

- Se enviará un disquete con el programa por participante, una fotocopia del DNI y una carta firmada declarando que todo el código, fuente y ejecutable, ha sido desarrollado por el participante sin emplear código de terceras personas. El que quiera puede enviar un segundo disquete como copia de seguridad del primero.
- El programa debe funcionar bajo DOS exclusivamente.
- El fichero ejecutable puede tener instrucciones de los procesadores 286 y 386 siempre y cuando el programa funcione en modo **real**, no en modo protegido.
- **El disco deberá tener el siguiente contenido:**
  - Directorio FUENTE, en el que estará el programa en código fuente. El archivo tendrá como nombre el D.N.I. del autor y las librerías o ficheros externos que emplee para poder recompilarlo.
  - Directorio EXE, con el programa compilado nombrado como DNIdelautor.EXE (por ejemplo: 14556435.EXE) y si lo necesita, el programa RUNTIME correspondiente. Además incluirá un archivo llamado DNIdelautor.TXT en el que se pondrá el nombre completo del participante y su teléfono de contacto (con prefijo de provincia).
- **El programa deberá cumplir las siguientes características y normas de funcionamiento:**
  - No puede hacer uso de memoria EMS o XMS, *ni acceder a direcciones absolutas de memoria.*
  - Podrá crear y utilizar un fichero auxiliar con el siguiente nombre: DNIdelautor.DAT.
  - Modificará el fichero TABLERO.DAT, creado por el árbitro, para anotar su movimiento.
  - Dispondrá de 12 segundos totales por jugada.
  - Comprobará el turno que le corresponda leyendo el TABLERO.DAT. Si está todo a 0, será el primero en jugar y su ficha será el número 1. Si encuentra sólo una ficha, le corresponderá el 2. Para conservar físicamente el turno o número de jugador puede grabarse en el fichero auxiliar. Otra manera de hacerlo consiste en comprobar que si la cantidad de fichas (unos y doses) del tablero es par o 0, es el jugador 1 y si es impar, el jugador 2.
  - Sólo hará un movimiento por turno, cerrará el fichero auxiliar y el TABLERO.DAT en el que previamente habrá introducido un 1 ó un 2 en la casilla deseada, y liberará la memoria RAM utilizada al acabar su ejecución.
- **Fichero TABLERO.DAT:** Tamaño 42 bytes. Fichero en formato binario que representa un casillero de 7 columnas por 6 filas, donde los 7 primeros bytes componen la fila superior de izquierda a derecha por donde entran las fichas. Los valores almacenados en este archivo son números decimales del 0 al 2 en formato binario.
- **Recursos disponibles:**
  - 512 Kb. como cantidad máxima disponible en disco para alojar el fichero temporal del participante.
  - 540 Kb. de memoria RAM convencional.
  - Acceso solamente para la lectura del reloj de la BIOS.
- **Causas de eliminación:**
  - Quedarse "colgado" (bloqueado), producir un error de ejecución o realizar un movimiento incorrecto.
  - Existir errores de lectura en los disquetes enviados.
  - Escribir en pantalla, leer de teclado, hacer pausas o acceder a cualquier otro recurso del sistema.
  - No cumplir todas las normas anteriormente expuestas.
  - Si en el archivo TABLERO.DAT se introducen valores que no sean 0(0000b), 1(0001b) ó 2 (0010b). Hay que tener cuidado para no introducir los valores 0, 1 y 2 en ASCII (códigos 48, 49 y 50 en decimal).
  - Funcionar en modo protegido.
- El plazo de recepción de los programas acaba el 15 de marzo de 1.995.
- Los resultados saldrán publicados a partir del número de marzo, en función del número de programas recibidos. En el disco de la revista se incluirá el código fuente de los cinco finalistas.
- Las personas que quieran que les sea devuelto su disquete incluirán en el sobre un segundo sobre con su dirección como destinario, y el franqueo en sellos necesario.
- SOLO PROGRAMADORES se reserva el derecho de utilización de todo el material enviado.
- Las personas que envíen el programa para participar aceptarán implícitamente las bases y desarrollo del concurso.

SEDE DEL TORNEO

Características del ordenador empleado para las partidas:
- Procesador: 486 DX2 50 Mhz
- Sistema operativo: MS-DOS 6.2.





NOTICIAS

3COM PRESENTA LA TECNOLOGIA PACE

La compañía 3Com presentó en la recientemente clausurada edición de Interop la tecnología PACE, en colaboración con otras siete empresas del sector informático. PACE es el acrónimo de Priority Access Control Enabled. Se trata de una extensión de Ethernet que permite utilizar las aplicaciones en tiempo real y multimedia, aprovechando para ello las infraestructuras existentes.

Sus creadores consideran esta tecnología como la evolución lógica de la estrategia High Performance Scalable Networking, ya que permitirá a los usuarios disfrutar de las aplicaciones interactivas, multimedia y en tiempo real más sofisticadas (vídeo-conferencias con PC, gestión de imágenes, mensajes de audio, distribución de la información urgente y control de procesos) en cualquier red Ethernet.

El resto de las compañías que realizaron el anuncio fueron: Apple, Dell, Novell, Oracle, Silicon Graphics, Starlight Networks y Sun. Todos ellos han presentado programas de soporte de la tecnología PACE.

SYBASE Y NOVELL TRABAJAN DE FORMA CONJUNTA EN LOS ENTORNOS DE TRABAJO EN GRUPO

Según el acuerdo al que llegaron en el pasado mes de octubre las compañías Sybase y Novell, ambas desarrollarán conjuntamente una estrategia orientada a los entornos de trabajo en grupo. El primer fruto de esta colaboración es el producto integrado SybaseWare. En respaldo al mismo se ofrecerán formación integrada y a medida, márketing, distribución y soporte.

SybaseWare incorpora el Sybase WorkGroup SQL Server 10 y los sistemas operativos de red NetWare y UnixWare de Novell. El objetivo es la simplificación de la instalación, de la administración y de la complejidad de la gestión de este producto. Se pretende facilitar así a los usuarios el desarrollo con el mismo, además de proporcionarle un importante ahorro en sistemas cliente/servidor.

Ya se puede disponer de Sybase WorkGroup SQL Server sobre el entorno NetWare. Se espera que se encuentre disponible sobre UnixWare en el primer trimestre del presente año.

LOS ROUTERS DE CISCO SYSTEMS SOPORTAN EL PROTOCOLO NLSP DE NOVELL

Cisco Systems Ibérica ha anunciado el soporte del protocolo NLSP (NetWare Link Services Protocol) de Novell en toda su gama de routers. El soporte a los protocolos NLSP e IPXWAN 2.0 se ha incorporado como funcionalidad estándar incluida en la versión 10.3 del sistema operativo de interconexión de redes (IOS) de Cisco (disponible desde el comienzo de 1995).

El protocolo NLSP ha sido desarrollado como un estándar industrialmente aceptado y soportado por los principales fabricantes de routers. Entre ellos se encuentra Cisco Systems.

El soporte a este protocolo permite a los clientes de redes Netware aprovechar las posibilidades ofrecidas por las comunicaciones de área extensa (WAN).

El Netware Link Services (NLSP) es una nueva tecnología de enrutamiento para redes basadas en IPX que libera ancho de banda para tráfico de datos. Sustituye a los protocolos RIP (Routing Information Protocol) y SAP (Services Advertising Protocol) de Novell. ■

LIBROS

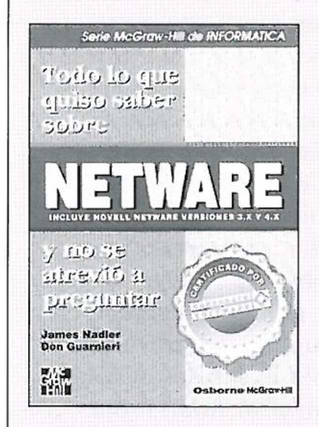
ASÍ FUNCIONA SU MAC... POR DENTRO



Si siempre quiso conocer los entresijos que se esconden tras el amigable aspecto de un Mac, ahora tiene la oportunidad de hacerlo de una forma sencilla y, como corresponde a la filosofía de Apple, muy intuitivamente. Basada en la serie Así funciona de la revista MacUser, esta obra explica cuidadosamente todos los aspectos de la tecnología que pone en funcionamiento los populares Macintosh. Profusamente ilustrado, este libro puede hacer que se comprendan los más complejos procesos, gracias a sus gráficos y esquemas temáticos.

Autor: John Rizzo y K. Daniel Clark
Editorial: Anaya Multimedia (91) 320 01 19
222 páginas
Precio:

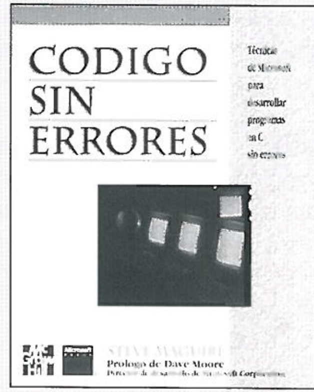
TODO LO QUE QUISO SABER SOBRE NETWARE Y NO SE ATREVIÓ A PREGUNTAR

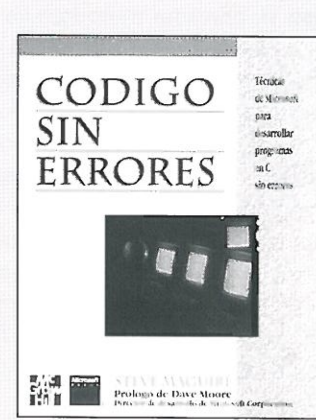


Los problemas más comunes que se le presentan al usuario de NetWare se han reunido en este pequeño y manejable volumen, abordándose de una forma sencilla y comprensible. Acudiendo a él se podrá dar respuesta a cuestiones tan frecuentes como: ¿Cómo conseguir que el servidor reconozca más de 16 Mbytes de memoria?; ¿Cuál es la diferencia entre unidades de red y unidades de búsqueda?; ¿Cómo impedir el uso de SALVAGE para recuperar un archivo problemático que se desea que permanezca borrado?; etc.

Autor: James Nadler y Don Guernieri
Editorial: McGraw-Hill (91) 372 84 09
212 páginas
Precio: 2.020 ptas.

CÓDIGO SIN ERRORES





En esta obra se han recogido los conocimientos de los especialistas de Microsoft para desarrollar programas en C sin errores. Para eliminar los errores del código, se intenta analizar el porqué de los mismos, y cómo podrían haberse prevenido o detectado de forma automática. Ya puede beneficiarse de la experiencia de Steve Maguire, así como de los múltiples ejemplos prácticos que el libro ha recopilado.

Autor: Steve Maguire
Editorial: McGraw-Hill (91) 372 84 10
229 páginas
Precio: 2.913 ptas.

IRONIAS

Cantaba Bob Dylan cuando aún había memorias de núcleos de ferrita: "El que se interponga en el camino puede salir herido, porque los tiempos están cambiando". Un día, Rosa, que trabajaba desarrollando transacciones para el mainframe de un gran banco, oyó el equivalente informático de esta canción y le entró el miedo. Le dijeron que COBOL, el lenguaje en que ella había nacido y en el que desarrollaba la totalidad de su trabajo, estaba en vías de extinción. Se lo decían sus amigos y lo leía en las revistas: abandona el barco o dedícate a repartir pizzas.

Esto ocurrió durante el reinado de Alfonso XII o así, no estoy muy seguro porque ando pez de Prehistoria; en todo caso hace más tiempo del recordable, por la era en que los diskettes de ocho pulgadas eran la revolución de los medios de almacenamiento.

Las amenazas de desaparición no han dejado de caer sobre COBOL desde entonces. Rosa, con los años, se ha hecho toda una gurú en el Centro de Proceso de Datos. Tanto ella como el resto de personas de su departamento siguen fieles a su querido, obsoleto y mantenible COBOL, hoy decano de los lenguajes de programación, un muerto que se obstina en respirar a pleno pulmón y dar trabajo a miles de desarrolladores.

Otra que tal: Cuando Niklaus Wirth, creador de Pascal, ya llevaba años intentando promocionar a Modula-2, lenguaje con el que pretendía superar las limitaciones de su predecesor, en mi Facultad la asignatura de programación se seguía impartiendo con Pascal, eso sí, con una metodología orientada a objetos, porque hay que estar en cabeza de la tecnología y tal. Ahora que Wirth lleva otros tantos años suplicando que inviten a las fiestas de sociedad a Oberon, el fruto definitivo de sus reflexiones y elegante lenguaje orientado a objetos, en mi Facultad se han puesto a cantar las excelencias de Modula-2. Fuera de las universidades es aún más divertido: lo único que vende un poco es Pascal, y en orientación a objetos, Pascal With Objects, una especie de extrapolación apócrifa de C++ con la que Wirth no quiere tener nada que ver. Me parece oírle gritando: "¡Pascal por aquí, Pascal por allá, siempre Pascal! Que me dejéis en paz al Pascal, jolín, que tengo yo un lenguaje nuevo que resuelve de una vez por todas las... Pero bueno, ¿me está alguien escuchando o qué?"

Está claro que unos lenguajes tienen éxito y otros no. Lo curioso es que, a menudo, esto ocurra en contradicción abierta con las capacidades o carencias del lenguaje en relación con sus competidores. A veces, incluso en contra de los deseos de quienes los concibieron.

Ahora que, si uno lo piensa bien, este fenómeno no tiene nada de exclusivo. Parece que la permanencia en el candelero de un sistema, una teoría, un producto, lo que sea, se consigue sólo cuando llega en el momento adecuado, gusta a un sector clave del público, y recibe los apoyos adecuados de gente con poder y/o mano izquierda. Si falla alguna de estas premisas, no hay nada que hacer. Y si no, que se lo pregunten a los inventores del sistema Betamax, a quienes les faltó la mano izquierda mercantil que demostraron los defensores de su rival VHS para llevarse el gato al agua con un sistema más aparatoso y más imperfecto. O a los fundadores de la Comuna de París, a quienes no habría venido nada mal la clase de obstinado apoyo oficial que aún sigue recibiendo el plúmbeo lenguaje ADA en los EE.UU. O a Van Gogh, que no se llevó ni un duro de royalties por sacar su API de gráficos antes de tiempo; y es que, como dijo Confucio, "tener razón demasiado pronto es como no tener razón". O al creador de FORTH, que puso en su lenguaje toneladas de simplicidad y elegancia, pero desde luego omitió darle ese toque de encanto marujil y populista que ha conducido a un engendro llamado BASIC al Olimpo de la programación.

En fin, que las reglas del éxito son demasiado caprichosas. Al final, lo único que le puede ayudar a uno es tener potra o buscarse un mecenas con posibles. De la excelencia técnica mejor olvidarse, ése no parece ser un factor. Ah, y si algún día le presentan en un cóctel a Niklaus Wirth, ni se le ocurra decir: "Encantado de conocer al genial creador de Pascal". Sería como preguntarle a Isabel Preysler por Julio Iglesias.





BIT MANAGERS,

Juegos para consola con sabor español.

En este número, SOLO PROGRAMADORES entrevista a BIT MANAGERS, una joven aunque experimentada compañía de programadores de videojuegos . En colaboración con otras empresas como Infogrames, han creado juegos como *Light Corridor*, *Asterix* o *Pitufos*.

Después de algunas desagradables experiencias en la empresa donde trabajaban, New Frontier, este grupo de *hacedores de juegos*, decidieron hacer una compañía dirigida por ellos mismos.

Esta ha sido una decisión complicada, porque el nombre de New Frontier ya tenía un determinado prestigio, y abandonarlo tenía sus riesgos, pero han elegido "volver a empezar" y forjar una imagen distinta para esta nueva aventura, sin renunciar a su historia, que a pesar de tener algún momento verdaderamente oscuro, es donde se hallan sus auténticas raices.

Experiencia en el desarrollo de juegos

Nuestra experiencia es de unos seis años haciendo videojuegos. Todos comenzamos en nuestra casa con un Spectrum, haciendo programillas, algún grafiquillo, musiquillas ... Todas estas "cosillas" sirvieron de base para hacer el primer "jueguecillo", llamado "Timeout", que apareció allá por 1989. Con esto adquirimos la experiencia necesaria para realizar proyectos más importantes. Primero fueron algunas conversiones de Spectrum a MSX, y después llegó el momento en el que afrontamos realizar juegos completos para Spectrum, Amstrad y MSX.

Programar ¿es una afición, un trabajo?, ¿vivís de esto?

Por supuesto, esto comenzó como una afición. Cada uno de nosotros hacía en su casa sus primeros pinitos con ordenadores, y poco a poco fuimos siendo conscientes de que este campo estaba cada día más en auge. Cuando empezamos a ver los primeros resultados de nuestro esfuerzo, nos dimos cuenta de que éste iba a ser nuestro trabajo. Actualmente, todos vivimos de esto, pero quizá lo más importante es que nunca ha dejado de ser nuestra mayor afición.

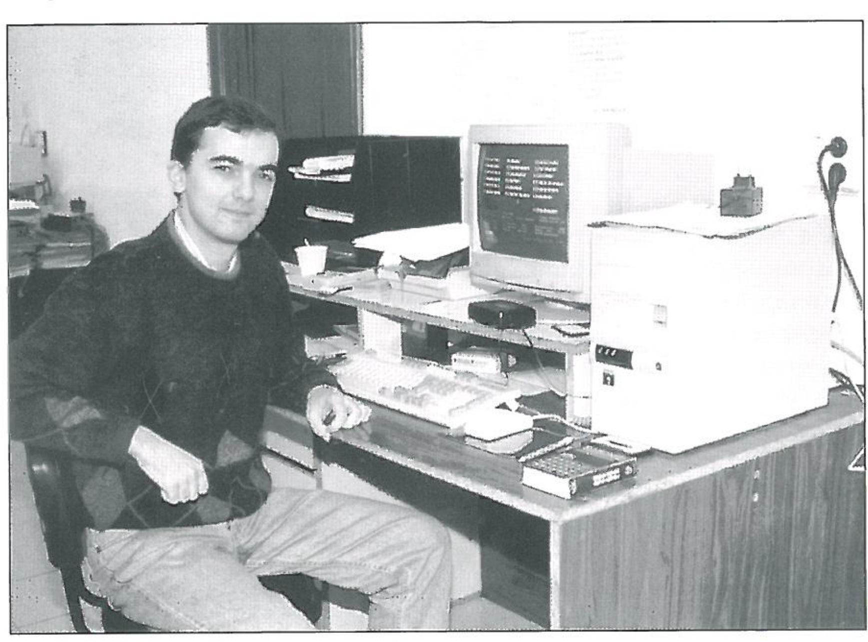
¿Cómo fueron vuestros comienzos y primeros pasos con Infogrames?

Como ya hemos dicho antes, nuestro primer proyecto fue "Timeout", para Spectrum. Luego hicimos las versiones para Amstrad y MSX, y después varias conversiones a MSX. Los primeros programas importantes que hicimos fueron a raíz de entrar en contacto con Infogrames. El primer proyecto que nos encargaron fue "Hostages", y simultáneamente hicimos "Magic Johnson". El resultado de "Hostages" fue magnífico, y nos abrió las puertas a la realización de nuevos proyectos ("Norte y Sur", "Light Corridor", "Mystical"). En esos momentos, el mercado estaba sufriendo la transición de los ordenadores a las consolas, e Infogrames volvió a confiar en nosotros. Hicimos "Pop Up", "Bomb Jack" y "Metal Masters" para Gameboy, después "Asterix" para Gameboy y NES y "Los Pitufos" para gameboy, NES, Game Gear y Master System, y finalmente los actuales proyectos en curso, "Obélix" para Gameboy y Súper

Nintendo y Tintín y Spirou para todas las consolas de 8 bits.

¿Hasta dónde llega vuestra creatividad?

Los últimos proyectos de Infogrames son unas licencias muy importantes. Las compañías propietarias de los derechos de "Astérix" y "Pitufos" son muy meticulosas con sus personajes, y quieren saber en todo momento que pasará en el juego. Por ello, en Infogrames realizan un diseño muy general sobre papel de cómo será el juego; sus fases, personajes, acciones, etc. Este es nuestro punto de partida; a partir de aquí añadimos todo lo necesario para convertirlo en un buen juego: diseño de los gráficos, banda sonora, jugabilidad, detalles técnicos, etc. Nuestra creatividad en los juegos es muy elevada, pero Infogrames supervisa todo el desarrollo del juego, cosa lógica puesto que ellos arriesgan mucho dinero en cada proyecto.



Condiciones con Infogrames

Las relaciones económicas entre ambas empresas son óptimas. Creemos que económicamente, estamos equiparados con los grupos de programación de más renombre, aunque esto siempre es algo difícil de comprobar, ya que éste es un tema del que no se suele hablar mucho al público en general.

Cual ha sido el trabajo que más os ha costado hasta ahora

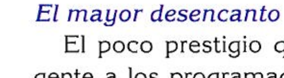
Quizá lo más complicado ha sido adaptarnos técnicamente a las mayores capacidades de las consolas. Los primeros gráficos de Spectrum, se hacían en Spectrum, pero en una Gameboy no se puede dibujar... Empezamos a hacer los gráficos de Gameboy con un Amstrad en modo 4 colores, pero como os podéis imaginar, era bastante laborioso. Finalmente, tuvimos que prepararnos una serie de "tools" (herramientas de programación) para las diferentes consolas, con más capacidad gráfica, e incluso ahora hemos tenido que adaptarlas para trabajar con la Súper Nintendo.

¿Y el que menos?

Seguir dando a nuestros juegos ese toque personal de calidad que nos es característico.

La mejor satisfacción

No podemos decir algo concreto que sea nuestra mayor satisfacción. Quizá lo que más nos satisface es la gran acogida que están teniendo nuestros juegos en toda Europa y sobre todo en España.



El mayor desencanto

El poco prestigio que da mucha gente a los programadores de videojuegos.

Algo de lo que estéis orgullosos

De haber resistido todo este tiempo en el mundo de los videojuegos, intentando siempre mantenernos en la cresta de la ola.

Alguna parida

Pese a no tener la llave para abrir la madera, y sin la experiencia empresarial necesaria, se hace lo que se puede y todavía estamos en ello. Pero sabemos que todo se arreglará cuando llegue lo del ayuntamiento. (Es una parida impresionante, pero nos haría mucha ilusión que lo publiquéis).

Equipos que utilizáis de desarrollo

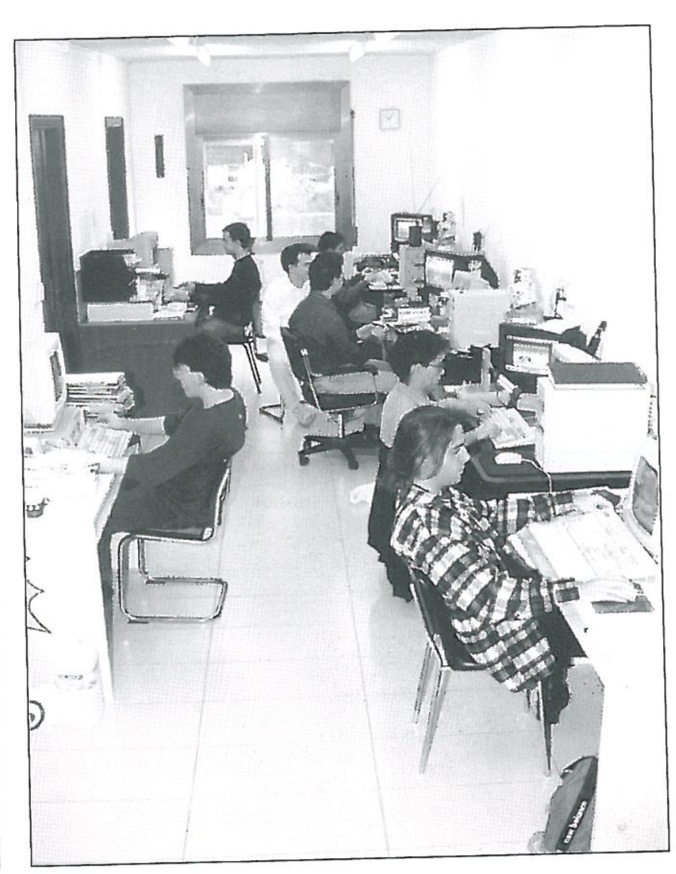
Después de haber pasado por todo lo imaginable (Spectrum, Amstrad, Atari), ahora todo lo hacemos con PC: programación, gráficos, diseño de las fases. Lo único que todavía hacemos con Spectrum es, aunque sea difícil de creer, la música. Hasta ahora ha sido suficiente por la similitud de los chips de sonido de las diferentes consolas de 8 bits, y el software que hemos preparado para Spectrum es bastante bueno, pero con los proyectos de Súper Nintendo utilizamos también el PC y teclados MIDI para la música.

Las herramientas de trabajo nos las hacemos nosotros mismos. Esto ha sido necesario para aprovechar al máximo la capacidad de las consolas y nuestro método de trabajo.

¿Cuánto se tarda en desarrollar un juego de consola?

Por supuesto, depende del tipo de juego, memoria y la consola. Un juego de Gameboy de 1 Megabit (por ejemplo Astérix) se hace en unos cuatro meses. Las versiones de ese mismo juego para otras consolas se hacen en unos tres meses, pues parte del trabajo se puede aprovechar. En cambio, un juego de Super Nintendo de 8 Megabits puede llevarnos más de seis meses.





¿Os costó adaptaros a la forma de programar consolas?

Como hemos dicho antes, fue una de las cosas que más nos costó. Afortunadamente, teníamos disponible toda la información técnica de las consolas antes de empezar a trabajar con ellas, y pudimos preparar las "tools" necesarias. El otro problema es la filosofía de los juegos para consola. Las mayores posibilidades de las consolas son también más difíciles de utilizar, y los programas llegan a ser técnicamente muy complicados.

Formación técnica de los componentes

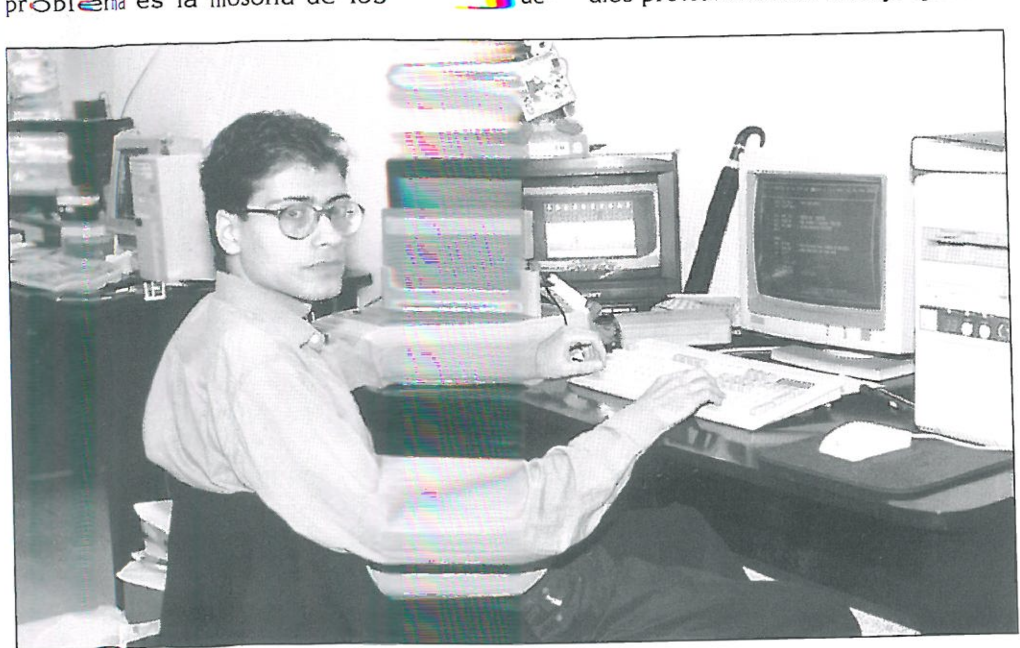
Ninguno de nosotros tiene estudios previos sobre el trabajo que desempeña. Podemos decir que todos somos autodidactas. En alguno sitios te enseñan a programar, pero de ahí a los videojuegos hay un abismo, el cual sólo se puede salvar con mucha dedicación y autoenseñanza. Mucho más importante que unos conocimientos en programación son las ganas y capacidad de aprender y una gran afición por los videojuegos.

¿Ensamblador y debugger o "C" como los seres racionales?

Según a qué nivel. Los juegos de consola se hacen en ensamblador, y ni siquiera podemos utilizar un debugger. En cambio las "tools" las hacemos en "C" y ensamblador combinados.

Seguro que habéis pensado en algún proyecto propio...

Tenemos muchas ideas de juegos que nos gustaría hacer. El problema es que hoy en día la competencia en el mercado de las consolas es muy dura, y para tener éxito es casi imprescindible una buena licencia ("Astérix", "Pitufos",...).





Consejos para programadores nóveles

Hoy en día es mucho más difícil empezar a programar que hace 6 años. Antes tenías tu ordenador y podías hacer tus propios programas, pero hoy en día te limitas a meter un cartucho en la consola y coger el joypad. La única posibilidad de programar algo es el PC o el Amiga, pero en estos ordenadores los juegos son cada vez más complejos y difíciles de hacer para un principiante. Es también muy importante haber jugado mucho para conocer las técnicas utilizadas por otros programadores más experimentados. Nuestra experiencia nos dice que es mejor aprender por nosotros mismos antes que ir a una academia o algo parecido.

A los programadores les recomendamos sobre todo que programen en ensamblador. Un programador en ensamblador puede aprender "C" fácilmente, pero no al contrario. Sería bueno que hagan algo consistente, casi un juego completo, no con muchas fases ni muy complicado, pero que tenga todo lo que hay en los juegos: enemigos, scroll de pantalla, puntuaciones, menú... Con esto puede presentarse en una compañía en la que le enseñen las técnicas actuales y le encarguen algo más complicado (por lo menos nosotros sí lo haremos).

A los grafistas les recomendamos que dibujen mucho y que se fijen mucho en los gráficos de videojuegos comerciales. Esta es la mejor manera de aprender. En principio pensamos que es mejor empezar haciendo gráficos a 16 colores y no complicarse con más, y que intenten practicar con gráficos de fondo y animación de personajes.

A los músicos les queremos decir que hoy en día es imprescindible aprender ensamblador, porque de esta manera pueden programar sus propias rutinas de música y conseguir que ésta suene exactamente como ellos deseen.

¿Programaréis hasta la muerte?

Programaremos hasta que dejen de existir los videojuegos. ¿Vosotros pensáis que algún día dejarán de existir? Nosotros no. ■

Ficha de los componentes:

- Isidro Gilabert
Edad: 24
Cargo: Jefe de Programación.

- Alberto José González
Edad: 22
Cargo: Músico y grafista.

- Rubén Gómez
Edad: 24
Cargo: Grafista y designer.

- Ricardo Fernández
Edad: 23
Cargo: Programador.

- Sergio Palacios
Edad: 21
Cargo: Grafista.

- Daniel López
Edad: 22
Cargo: Programador.

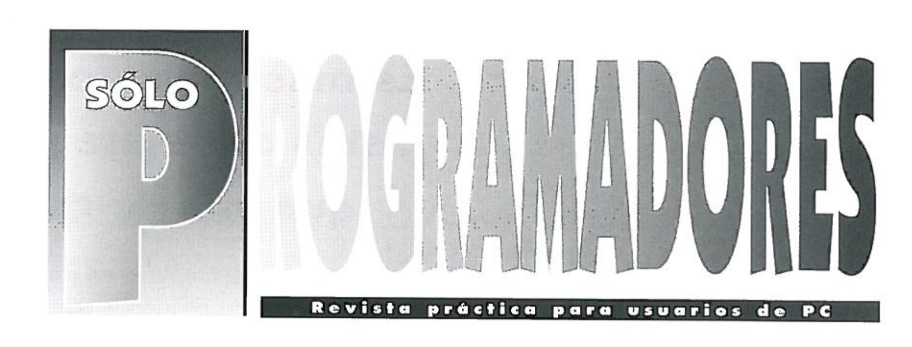
- Alberto Pleguezuelos
Edad: 25
Cargo: Grafista.

CORREO LECTORES

Para cualquier consulta relacionada con la programación o sobre los temas de los ártículos que aparecen en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A



Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

Deseo suscribirme a la revista SOLO PROGRAMADORES acogiéndome a la siguiente oferta:

1 año + 1 regalo (Filtro monitor) **por sólo** 10.995 ptas. Estudiantes de carreras técnicas 40% descuento: 8.250 ptas.

Nombre y apellidos...Domicilio...Población..........................

Provincia..C.P ....Fecha de nacimiento...............................Profesión...

FORMA DE PAGO:

Nº Teléfono ....

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Contra-reembolso del importe más gastos de envío.

☐ Giro Postal (adjunto fotocopia del resguardo).

☐ Con cargo a mi tarjeta VISA nº ....

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto...

☐ Domiciliación bancaria.

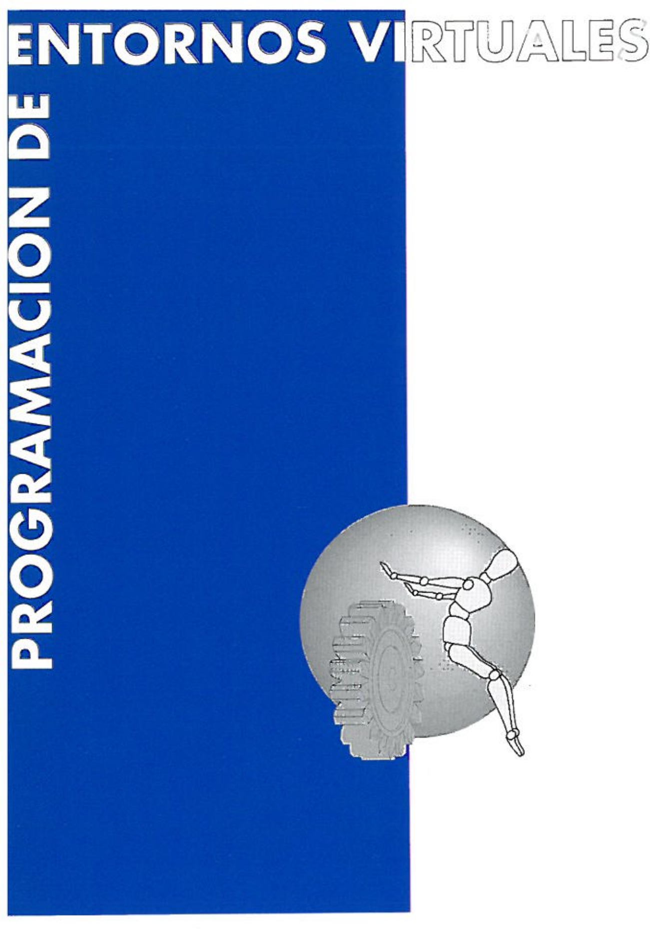
Señor Director del banco .... Nº Entidad ....

Nº Agencia .... Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número...

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.



OCULTACION DE OBJETOS

Luis Fernando Fernández

Cuando se trabaja con planos y estos son tratados de manera individual, a la hora de ser dibujados en pantalla surge el problema de que se pueden superponer inadecuadamente, apareciendo en pantalla planos que no se deberían ver. Existen multitud de métodos para evitar este problema, algunos de éstos van a ser comentados en el presente artículo.

La primera división que se podría hacer sobre los distintos métodos se basa en el momento de aplicarlos, si se puede realizar antes de proyectar los objetos sobre la pantalla de una manera eficiente, conviene hacerlo, por que de este modo se ahorran cálculos, incluso si no contempla todos los casos, ya que después se puede aplicar otro algoritmo más preciso para terminar eliminar las superficies que no se deben ver. Los que se aplican después de proyectar los planos, deben almacenar la profundidad a la que estaba cada plano antes de ser proyectado, ya que si no se dispone de dicha información es imposible deducir el orden en que deben dibujarse los planos.

Además se puede distinguir entre aquellos que trabajan directamente con los polígonos y los que lo hacen con objetos para ver cual está delante y cual detrás, éstos últimos se suelen aplicar en primer lugar para minimizar el número de polígonos que hay que ordenar (la mayoría de métodos utilizan algoritmos de ordenación, los cuales son generalmente lentos, por tanto cuantos menos polígonos haya que ordenar, mejor, y siempre es más rápido ordenar dos listas de veinte polígonos que una de cuarenta, por eso conviene primero ordenar los objetos, para después ordenar los polígonos de cada objeto).

Es más rápido ordenar dos listas de veinte polígonos que una de cuarenta

De lo dicho se puede deducir que no existe el algoritmo perfecto, esto es, que elimine las superficies ocultas rápidamente y por tanto de lo que se trata es de elegir varios métodos que se complementen lo más eficientemente posible.

A continuación se van a explicar en detalle los más conocidos, analizando sus pros y sus contras, dificultad para ser implementados, posibles fallos, etc.

METODO DE PREORDENACION

Como su nombre indica, consiste en ordenar los polígonos a la hora de introducirlos en la base de datos, para asegurarse que en tiempo de ejecución no aparezcan incorrectamente dibujados.

Además se eliminan de la base de datos aquellos polígonos que no se pueden ver desde la posición en la que se encuentra el observador (ver figura 1). Por ejemplo, en el caso de un cubo, se eliminarían en el peor de los casos tres de los seis planos, ya que es imposible ver a la vez más de tres de sus planos, y en el mejor de los casos, tan sólo se vería uno de los planos. Para comprobar lo dicho, basta con coger un dado y girarlo en el espació, se verá que no hay manera de ver cuatro o más caras a la vez.

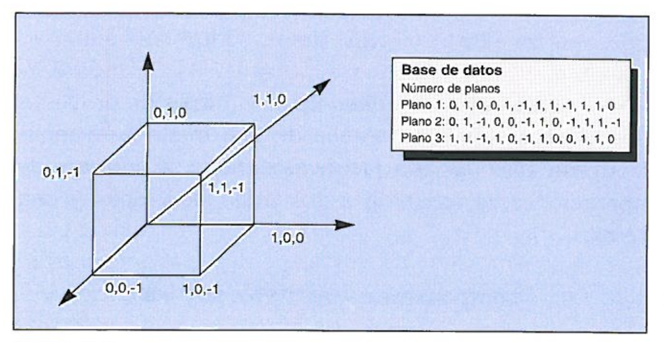


FIGURA 1.

El principal inconveniente es la limitación del movimiento, ya que si se da plena libertad de movimientos al observador, al cambiar el ángulo de visión, no servirían los datos de dicha base de datos con lo que el objeto aparecería incorrectamente en pantalla. Es evidente que las caras que se ven no son las mismas si se mira al dado desde la derecha que desde la izquierda y por tanto la base de datos tampoco es la misma.

Se podría corregir ese fallo aumentando la base de datos, con las distintas ordenaciones de los polígonos que componen el objeto y que estas se fuesen seleccionando en función del ángulo de visión del observador, pero a medida que aumentasen los objetos, tanto en número como en complejidad, el trabajo se haría más pesado y llegaría un momento en que no se podrían contemplar todos los casos posibles. Para darse cuenta de que manera se dispara, lo mejor es seguir con el ejemplo del dado por tratarse de un objeto simple. Como se dijo antes, se pueden ver a la vez una, dos ó tres de sus caras y para cada uno de los casos habría que hacer la siguiente subdivisión:

Una cara: se puede ver desde seis posiciones distintas, una por cada cara, por tanto hay que almacenar las seis junto con los correspondientes ángulos desde los que se ve cada una para la posterior comprobación.

Dos caras: en este caso hay una posición por cada lado del dado, en total el número de posiciones es doce, pero el rango de ángulos desde los que se ve cada par es más difícil de delimitar y cada posición de la base de datos es mayor al tener que guardar la información de dos caras en vez de una como en el caso anterior.

Tres caras: el número de posiciones para este último caso sería igual al número de vértices, esto es, ocho, con tres polígonos por cada posición.

En total habría que guardar veintiséis posiciones, cada una de ellas conteniendo entre uno y tres polígonos, a parte del maremagno de ángulos que aparecerían para dilucidar que posición es la correcta en cada momento.

Como se puede observar, para un sistema real de tres dimensiones, es imposible utilizar este método debido a que si para un objeto tan simple como el dado se dispara la base de datos, almacenar los datos de una nave espacial medianamente decente, puede convertirse en una autentica odisea, y no precisamente espacial.

Habrá gente que después del ejemplo habrá pensado que no es necesario almacenar toda la información, que basta con almacenar cada polígono una vez junto con el rango de ángulos desde los que se puede ver, con lo que tanto el trabajo de creación como la posterior interpretación de la base de datos se simplifica. Bueno, esto es cierto para poliedros convexos (se explicó lo que eran en el número anterior) como en el caso del dado, pero para objetos más complicados no basta con almacenar las caras que se ven desde una determinada posición, además hay que guardarlas en un orden concreto, y si sólo se dispone de información de cada plano por separado no se sabe que plano está delante y cual está detrás (ver figura 2).

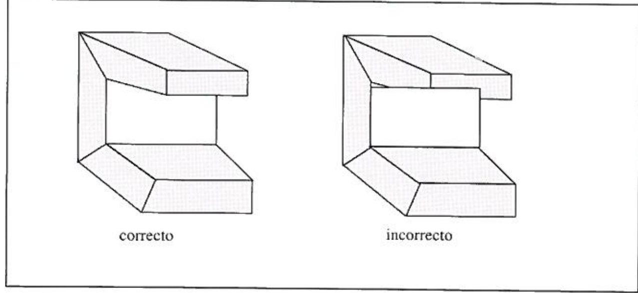


FIGURA 2.

Aun así no todo son inconvenientes, se puede utilizar en juegos de coches, debido a que siempre se ven los objetos de frente, el único inconveniente está en que habría que almacenar dos veces cada objeto en la base de datos, una para cada lado de la carretera ya que el ángulo de visión varía, y por tanto los polígonos que se ven y el orden en que se pintan. Aún así sería rentable su uso en este determinado tipo de juegos, ya que al no almacenar los planos posteriores, la base de datos quedaría bastante reducida.

La implementación del algoritmo es muy simple, pero no se va a realizar en esta serie debido a las limitaciones que impone su uso. Además no tiene mucho sentido utilizarlo en un simulador que permita plena libertad de movimientos, como es el caso del simulador que nos ocupa.

METODO DE LA NORMAL

Fue tratado en el artículo anterior, pero se enfocó desde el punto de vista del plano ya proyectado en pan-

talla, ahora se explicará como utilizarlo únicamente con los puntos en el espacio, o más exactamente con los tres primeros puntos de cada polígono después de transformarlos en función del punto de vista (situarlos en el espacio, trasladarlos y rotarlos). De este modo se gana en velocidad, ya que el número de cálculos previos a la comprobación de si el polígono está oculto, es mucho menor que en el algoritmo del número anterior. Cuando el polígono está oculto, en el peor de los casos, si se trata de un triángulo, se ahorrará la proyección de los tres puntos, y la posterior comprobación de cual de ellos es el de inicio del algoritmo de ocultación y el número de operaciones es similar al de la comprobación de las pendientes que se realizaba antes. Si aumenta el número de vértices del polígono, también aumentará el rendimiento con respecto al antiguo algoritmo.

Se basa en las propiedades del producto vectorial. Supongamos que los tres primeros puntos del polígono que se está comprobando son el A(x0,y0,z0), el B(x1,y1,z1) y el C(x2,y2,z2). En primer lugar se obtienen los vectores con los que posteriormente hay que trabajar, estos son dos, el primero es el que va del punto B al punto A, mientras que el segundo va del punto B al punto C (ver figura 3).

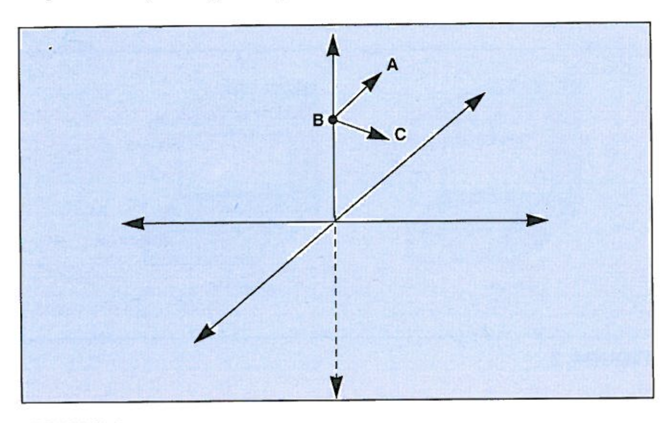


FIGURA 3.

Por definición, el producto vectorial de dos vectores es perpendicular a cada uno de dichos vectores y su sentido viene dado por la regla de la mano derecha, también conocida por regla del tornillo (ver figura 4). Consiste en colocar la mano derecha como si se fuese a estrechar la mano, con todos los dedos unidos excepto el pulgar que está perpendicular a los demás, a continuación se cierra la mano simulando el recorrido del vector inicial al final, cuando termina la operación el dedo pulgar indica el sentido del vector que resulta del producto vectorial.

Se debe multiplicar el primer vector, por el segundo, del vector resultante tan sólo interesa el signo de la componente z para este método, si es negativo el polígono se ve y por tanto se pinta mientras que si es positivo se descarta y se pasa a la siguiente comprobación.

Las fórmulas del producto vectorial para los vectores a y b quedarían:

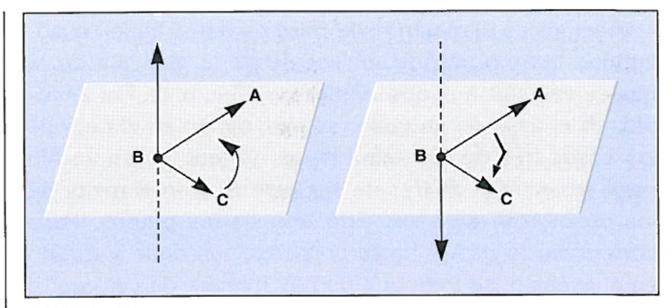


FIGURA 4.

$$a \times b = \begin{vmatrix} Ya & Za \\ Yb & Zb \end{vmatrix} X - \begin{vmatrix} Xa & Za \\ Xb & Zb \end{vmatrix} y + \begin{vmatrix} Xa & Ya \\ Xb & Yb \end{vmatrix} Z$$

Como sólo interesa la componente z, el número de operaciones se reduciría a dos multiplicaciones y una resta:

$$\text{componente } z = xa * yb - ya * xb$$

Esta fórmula viene de desarrollar el tercer determinante. En realidad sobra incluso la resta, por que como lo que interesa es el signo basta con comprobar los resultados de ambas multiplicaciones y si el primero es mayor el signo es positivo y en caso contrario es negativo.

Para que quede claro lo mejor es un ejemplo concreto. Supongamos que se tienen los puntos ya trasladados y rotados A(0,1,5), B(3,7,4) y C(4,2,5).

Para obtener el vector a = BA hay que restar a las coordenadas de A las de B con lo que se obtiene:

$$a = -3x - 6y + z$$

Si ahora se realiza la misma operación para hallar el vector b = BC se obtendrá:

$$b = x - 5y + z$$

Ahora tan sólo queda aplicar la fórmula para hallar la componente z, que es la que se necesita para ver si se pinta ó no se pinta:

$$\text{componente } z = xa * yb - ya * xb$$

Donde xa = -3, ya = -6, xb = 1 e yb = -5. Sustituyendo:

$$\text{componente } z = -3 * -5 - -6 * 1$$

Resolviendo se obtiene:

$$\text{componente } z = 21$$

Como el resultado ha sido positivo el polígono no se ve y por tanto queda descartado y no hay que pintarlo.



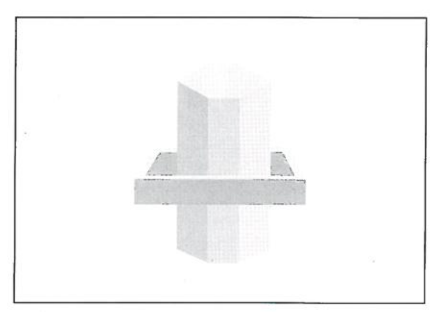


FIGURA 7.

Tiene el inconveniente de que no funciona cuando un objeto puede atravesar a otro objeto (ver figura 7). Claro, que no suele ser habitual, pero para evitar problemas de este tipo no se ha incluido en el listado del presente artículo, aunque probablemente se incluya en el siguiente.

METODOS DE ORDENACION

Como se puede observar, la mayoría de métodos requieren de algoritmos de ocultación, de todos modos, como ya se vieron distintos métodos y se compararon entre sí, en un artículo anterior del curso de técnicas de programación, no se van a volver a repetir ahora, tan sólo se va ha explicar el sistema que se ha usado en el programa ejemplo y el porque se ha elegido ese método, ya que a primera vista puede parecer un tanto ineficaz.

Inicialmente se tiene un array del que se conoce el número de posiciones ocupadas, y en cada posición están los datos del polígono, coordenadas en pantalla, color, etc. y la profundidad, que va a ser la que sirva como referencia para la supuesta ordenación, que como se verá no es tal, ya que consiste en recorrer la lista comprobando que posición tiene la z mayor, cuando se llega al final de la lista se cogen los datos del polígono que está en la posición seleccionada y se pinta en pantalla, poniendo a cero su coordenada z para que no vuelva a salir elegido, a continuación se repite el proceso hasta que se hayan pintado todos los polígonos en pantalla.

Por tanto la subrutina que se encarga de todo el proceso se compone de un simple bucle de tipo FOR que se repite tantas veces como elementos haya en la lista, dentro de dicho bucle se encuentra otro de iguales características que se encarga de recorrer el array, con una sentencia de tipo IF...THEN...ELSE para averiguar cual es el polígono que está más lejos en cada momento.

Como el número de operaciones a realizar en el interior de la subrutina se reduce a una simple comparación, ésta funciona rápidamente, aunque pueda parecer que es ineficaz, y el objetivo es conseguir rutinas rápidas, no bonitas si se quiere un movimiento fluido.

Si el número de polígonos a ordenar fuera desmesurado, entonces convendría buscar un método mejor, pero para el número de polígonos con el que se puede trabajar en sistemas de tiempo real sobre plataforma PC, este sistema es lo suficientemente útil.

PARA FINALIZAR

Los objetos deben ser poliedros, esto implica que ninguno de sus planos puede prolongarse ni hacia dentro ni hacia fuera de la estructura (ver figura 8).

Aunque esto pueda parecer algo obvio no lo es tanto, ya que se han hecho pruebas pidiendo a varias personas que diseñasen naves y este ha sido uno de los errores más comunes, ya que la mayoría prolongaba los planos hacia dentro dando por supuesto que estarían tapados por otros planos. De todos modos este fallo se puede solventar cuando se utilice un método de ocultación mejor, como puede ser el de árboles BSP, que será explicado en el próximo artículo.

Otro de los errores más habituales, y que no dependen de la librería para resolverlos, sino del encargado de diseñar las naves consiste en introducir polígonos que no tienen todos sus puntos en el mismo plano, y que por tanto a la hora de ser proyectados pueden dar lugar a polígonos cóncavos. Como la rutina de pintar polígonos da por supuesto que todos los polígonos que llegan a ella son convexos, cuando aparece uno cóncavo puede pasar cualquier cosa, ya que no tiene ninguna manera de detectarlo, apareciendo en pantalla polígonos sin sentido ó incluso quedándose bloqueada en algún bucle sin salida.

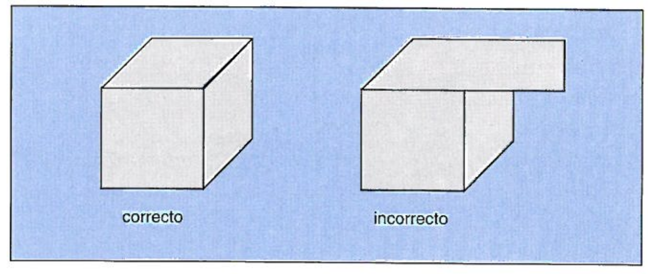


FIGURA 8.

Una manera de asegurarse de que los polígonos son correctos es trabajando únicamente con triángulos, ya que por definición hay un y sólo un plano que contiene a cada polígono por separado (en matemáticas, tres puntos definen un plano).

Cuando se trabaje con polígonos que posean más de tres vértices conviene tener cuidado, y en caso de que aparezcan fallos en pantalla comprobar cada uno de los polígonos del objeto, ó bien si no se descubre el fallo directamente, se podría implementa una rutina para comprobar si, dados un determinado número de puntos, estos pertenecen al mismo plano.

EL MES QUE VIENE

En el próximo artículo se terminarán de analizar los métodos de ocultación, con la explicación del método de los árboles BSP. Además se explicarán algunos efectos de luz para tratar de ambientar un poco más la simulación, e intentar hacer que las naves sean un poco más realistas, pero eso será dentro de un mes. ■

PROCEDIMIENTOS

Emilio Postigo

00101010010
00101010010
10101001010
01010101001
01010101010
101000100111
10101010101
00101101010
110101101010

El empleo cuidadoso de procedimientos, imprescindibles cuando se programa con lenguajes de alto nivel, debe ser tenido muy en cuenta por la persona que estudia ensamblador, ya que con su ayuda se pueden codificar con simplicidad y elegancia programas que, sin ellos, serían prácticamente irrealizables.

En el artículo anterior se vio la forma de codificar estructuradamente sentencias de control de programa mediante el uso de las directivas condicionales. Ahora bien, para que la codificación de un programa ensamblador resulte completa es preciso disponer de recursos equivalentes a los que permiten ejecutar un fragmento de código el número de veces deseado y desde cualquier punto del programa en un lenguaje de alto nivel, como son las subrutinas y funciones. En ensamblador es posible construir y utilizar estos recursos mediante directivas como PROC y ENDP e instrucciones como CALL y RET.

En este primer artículo sobre procedimientos se explorarán y repasarán estos elementos del lenguaje con el fin de proporcionar una visión completa del proceso de llamada a una subrutina y conseguir que el diseño y codificación de éstos pueda hacerse, hasta cierto punto, de forma automática.

Todo programa posee una zona de almacenamiento temporal

PROCESOS NEAR Y FAR

Cuando se construye un procedimiento la primera cuestión que ha de tener en cuenta el programador es el tamaño final del código de su ejecutable. Si la parte correspondiente a instrucciones no ocupa más de 64 Kb (modelos de memoria TINY, SMALL, COMPACT o FLAT), éstas se agrupan dentro del mismo segmento y se diferencian unas de otras, por así decirlo, en el valor del desplazamiento con respecto al inicio de segmento. En cambio, cuando las instrucciones ocupan más de 64 Kb (modelos MEDIUM, LARGE o HUGE), se necesitan dos valores para identificarlas: el inicio del segmento en el que se encuentran y la distancia (desplazamiento) de la instrucción con respecto al inicio de ese segmento, es decir, su dirección física.

El hecho de trabajar en unos casos con desplazamientos (datos de dos bytes) y en otros con direcciones (datos de cuatro bytes), obliga generalmente al programador a emplear diferentes instrucciones y directivas al referirse tanto a las instrucciones como a los datos de un programa. En el caso que nos ocupa esto se traduce

en el empleo de procedimientos NEAR (cerca) cuando trabajamos con código de tamaño inferior a 64 Kb y FAR (lejos) en caso contrario.

El presente artículo abordará principalmente la construcción y funcionamiento de procedimientos situados en el mismo segmento que el resto del código, es decir, NEAR, haciéndose aclaraciones en el momento oportuno para comprender qué ocurriría en el caso equivalente de un procedimiento FAR.

ACLARANDO CONCEPTOS

Para comprender la construcción y el funcionamiento de cualquier mecanismo no hay nada como percatarse de la necesidad de todos y cada uno de los elementos que lo constituyen. Dado que la llamada a un procedimiento y su ejecución pueden considerarse un mecanismo muy, muy delicado, nos plantearemos la construcción de uno cualquiera para entender el por qué de las instrucciones CALL y RET y su funcionamiento interno. Una vez comprendidos los conceptos implicados en su manejo, el resto será, casi, coser y cantar.

Supóngase, pues, que al codificar un programa se decide agrupar un conjunto cualquiera de instrucciones mediante las directivas PROC y ENDP tal y como se muestra en el listado 1, que NO finaliza con la instrucción RET, y se desea invocar ese fragmento desde distintos puntos del programa principal.

```
Bloque  PROC NEAR
        mov     SI, BX
        mov     AX, [SI]
                I
                I
                I
        mov     [SI+2], AX
Bloque  ENDP
```

IR Y VOLVER

Si queremos que el programa ensamblador funcione como uno escrito en un lenguaje de alto nivel en lo que se refiere a la ejecución de ese bloque de código, ha de ser posible llamarlo desde cualquier punto del programa principal y reanudar la ejecución de éste en la instrucción siguiente a la que efectuó la llamada cuando finalice la ejecución del procedimiento, como se esquematiza en la figura 1.

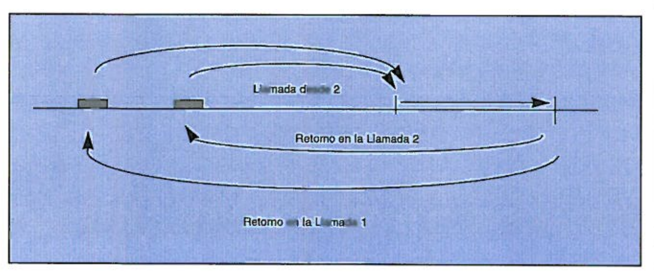


FIGURA 1.

En un principio se puede pensar que la ya conocida instrucción de salto incondicional JMP soluciona nuestro problema. En efecto, basta escribir JMP Bloque cuando sea necesario para transferir el control a la primera instrucción de las que forman el procedimiento. En este momento el conjunto de instrucciones se ejecuta, finaliza y se debe volver al módulo principal. Ahora bien, si el procedimiento ha podido ser llamado desde diferentes puntos, ¿cómo puede saber el procesador en qué instrucción prosigue el programa?

La pila se llena de los desplazamientos altos a los bajos

A estas alturas ya se puede suponer que el quid de la cuestión se halla en la instrucción RET. En efecto: RET es capaz de transferir el control a la instrucción precisa porque "supone" que el desplazamiento de ésta ha sido almacenado en un lugar muy concreto. Por lo tanto, su hipotético uso combinado con JMP lleva al programa a la catástrofe, dado que esta instrucción se limita a transferir el control y no se ocupa de nada más. Para que el conjunto funcionara sería necesario que JMP realizara dos tareas distintas: guardar en algún lugar predeterminado el desplazamiento de la instrucción siguiente (para que RET pudiera usarlo más tarde), y después saltar a la primera instrucción del procedimiento.

Como ya esperábamos, esto es exactamente lo que hace la instrucción CALL, de manera que ahora se puede describir el proceso correcto de llamada a Bloque de esta forma: la instrucción CALL Bloque almacena el desplazamiento de la instrucción siguiente a ella misma y después transfiere el control a la primera instrucción de la subrutina. Cuando se ha ejecutado todo el bloque de código, RET se encarga de tomar ese desplazamiento almacenado por CALL y de transferir el control a la instrucción adecuada.

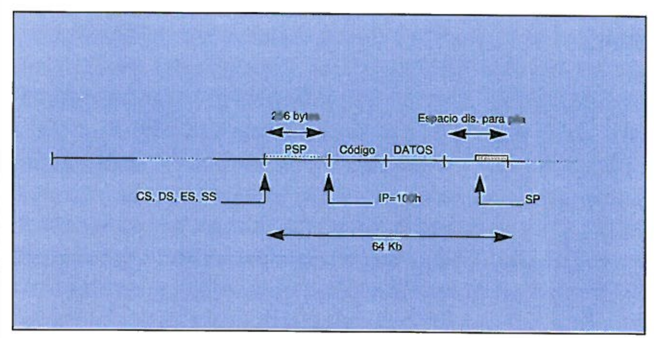


FIGURA 2.

De todo esto surge, inevitablemente, la cuestión de dónde y cómo se almacena el desplazamiento de la instrucción siguiente a la que efectúa la llamada. Para responderla de manera adecuada detengámonos un momento y fijémonos en la figura 2, en la que se muestra el esquema de un archivo COM una vez cargado por el sistema operativo.

LA PILA ES UN ALMACEN TEMPORAL

Recordemos que cuando el archivo COM es cargado para su ejecución, los registros de segmento, y entre ellos SS (registro del segmento de pila), son inicializados con el mismo valor: el de la primera dirección de segmento libre que encuentra el sistema operativo. El registro IP se inicializa con el valor 100h (256) y el registro SP (registro puntero de pila) con FFFEh. Por lo que respecta al código y a los datos, éstos se distribuyen a partir del PSP, desplazamiento 100h, llegando a ocupar el conjunto (256 + Tamaño del Código + Tamaño de Datos) bytes en total. Por lo tanto, a no ser que el espacio ocupado por todo lo anterior sea de 64 Kb, la última parte del segmento quedará "vacía". Pues bien, esta zona de la memoria, direccionable por las instrucciones del programa pero libre tanto de datos como de código, es a lo que se denomina pila, y se emplea como zona de almacenamiento temporal de datos. La localización y tamaño en cada programa de este almacén depende fundamentalmente del modelo de memoria escogido al codificar (SMALL, MEDIUM, etc), pero su manejo es el mismo en cualquier caso.

CALL FAR almacena en la pila la dirección de la instrucción siguiente

Comprobaremos al avanzar en el estudio de los procedimientos que el manejo de la pila posee un papel fundamental (pase de parámetros, definición de variables locales,...), pero de momento lo que nos preocupa es encontrar explicación al funcionamiento de CALL y RET. Una forma de comprender bien el funcionamiento de estas instrucciones es editar con el comando DEBUG el programa EJEMPLO1.COM y ejecutarlo paso a paso.

Al teclear U100,12c una vez cargado el programa se visualiza el código resultante del proceso de ensamblado del archivo EJEMPLO1.ASM. La primera instrucción es CALL 011F y viene seguida por la instrucción MOV AH, 02, situada a partir del desplazamiento 0103. Ejecutemos el comando T y veamos los resultados. Como podíamos esperar el registro IP presenta el valor 011F, y la siguiente instrucción que se puede ejecutar es MOV AX, 0600. ¿Se han producido más cambios? Fijémonos en el registro SP: antes de ejecutarse la instrucción CALL almacenaba el valor FFFE, y ahora guarda FFFC; es decir, ha sido decrementado en dos unidades.

La pila es una zona de almacenamiento que posee una peculiaridad: se va llenando de los desplazamientos altos a los bajos, usándose SP para apuntar al último byte ocupado dentro de la misma. Un decremento de SP equivale a un aumento de la información almacenada en la pila, mientras que un incremento de este registro indica que la pila se ha vaciado en parte.

Ahora se puede entender el sistema que emplea la instrucción CALL para almacenar el "camino de vuelta" antes de transferir el control a la primera instrucción de la subrutina: decrementa SP en dos unidades y almacena en la pila el desplazamiento de la instrucción que le sigue. La figura 3 esquematiza este hecho. Ejecutando DSS:FFFC comprobaremos que las posiciones de desplazamientos FFFCh y FFFDh con respecto al inicio del segmento de pila almacenan el valor 0103h, el desplazamiento de MOV AH, 02, como 03 en la posición FFFC y 01 en FFFD.

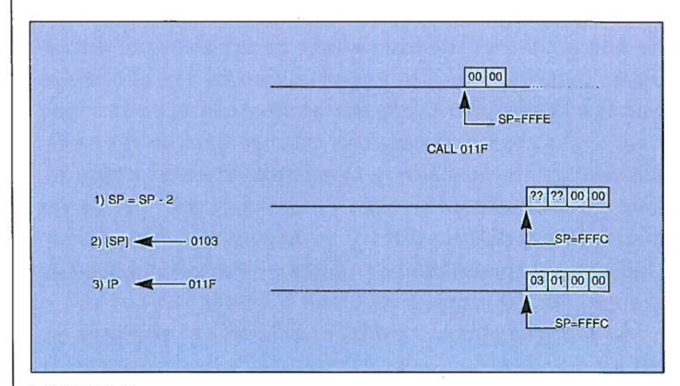


FIGURA 3.

Reanudemos la ejecución. Mediante las instrucciones que van del desplazamiento 011F al 012A la pantalla se borra (recuérdese que es muy conveniente ejecutar las interrupciones con el comando P), la subrutina se puede dar por finalizada y sólo resta ejecutar RET para volver al módulo que la llamó. Ejecutamos T y, en efecto, la siguiente instrucción que se puede ejecutar es MOV AH, 02. ¿Qué ha pasado con SP? Pues que ha recuperado el valor que tenía antes de la llamada al procedimiento, ocurriendo lo siguiente: la instrucción RET ha "supuesto" que en la posición cuyo desplazamiento está en SP se encuentra almacenado el desplazamiento de la instrucción de vuelta al módulo principal, por lo que ha tomado el contenido de esa posición y de la siguiente y lo ha depositado en IP, sumándose dos unidades a SP posteriormente. La figura 4 ilustra estos pasos.

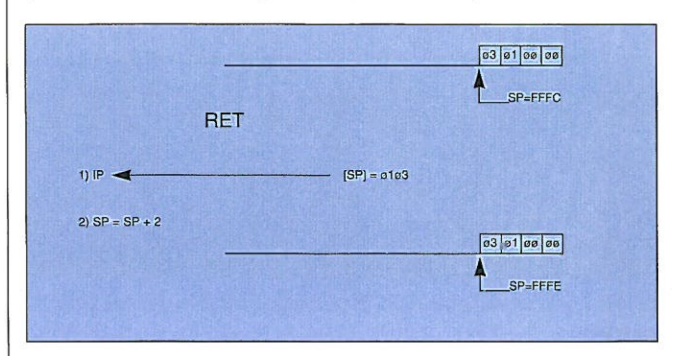


FIGURA 4.

¿Qué habría ocurrido si SP hubiera tomado un valor distinto de FFFCh en el momento de ejecutarse la instrucción RET? Recordemos lo visto en el artículo apare-

cido en el número 2; evidentemente no se hubiera retornado a la instrucción correcta, salvo una afortunada casualidad, y los resultados hubieran sido cuanto menos curiosos. Algo que ha de tenerse muy presente es que entre la primera instrucción de la subrutina y RET la pila NO debe sufrir un crecimiento o disminución netos o, lo que es lo mismo, todo proceso de aumento de esta zona de almacenamiento ha de estar compensado con la correspondiente disminución. De hecho, la llamada a la interrupción 10h (interrupción de vídeo) altera temporalmente el valor de SP, pero éste se encuentra convenientemente restaurado cuando se ejecuta la siguiente instrucción.

CASO DE UNA LLAMADA FAR

Si la llamada hubiera sido de tipo FAR, la instrucción CALL hubiera debido almacenar en la pila la dirección física completa de la instrucción siguiente, en el orden segmento-desplazamiento, decrementándose SP en cuatro unidades. Para compensar esto, la instrucción final del procedimiento, RETF, extrae esos cuatro bytes y los deposita en CS:IP, recuperando el registro SP el valor que poseía antes de la llamada. En las figuras 5 y 6 se muestran estas acciones, para las que se ha supuesto la ejecución de la instrucción CALL FAR 358F:011F, situada a partir de la dirección física 202B:0100.

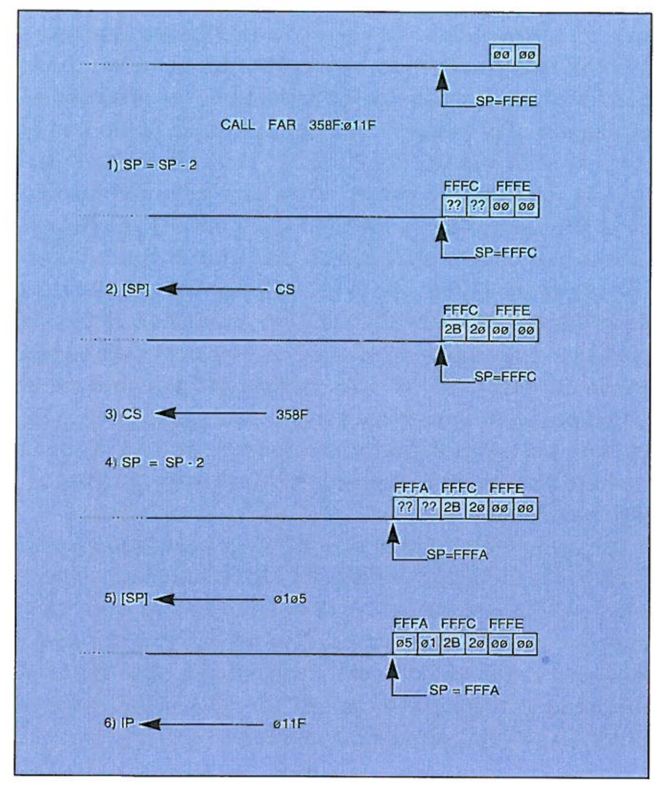


FIGURA 5.

EL PROCEDIMIENTO ES UNA "CAJA NEGRA"

Otro detalle que tiene presente el programador cuando diseña una subrutina es que su manejo no debe ser complicado. Esto quiere decir que su uso ha de ser in-

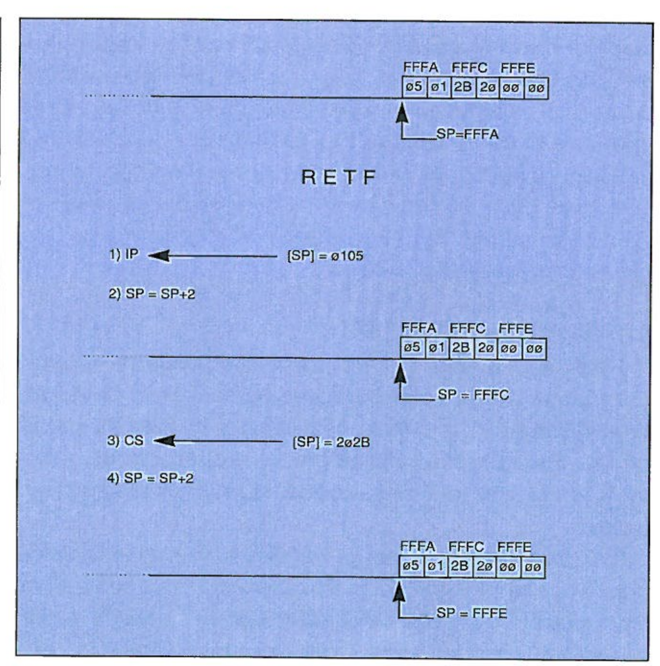


FIGURA 6.

mediato, que la subrutina va a efectuar la tarea que se le ha asignado modificando, como mucho, algunas variables conocidas, pero dispensando a su usuario de conocer a fondo el código de la misma. La subrutina es una "caja negra" que, a ojos del programador, realiza una labor concreta sin alterar el entorno del programa más allá de lo que se le solicita.

Una subrutina no debe modificar los registros que emplea

Si nos fijamos en el procedimiento empleado para borrar la pantalla en el programa EJEMPLO1.COM, podemos observar que después de su ejecución algunas variables del programa han experimentado un cambio: los registros AX, BX, CX y DX han perdido el valor que tenían antes de la llamada, por lo que la subrutina no se ha comportado como una caja negra. Si bien en este caso esto no tiene mayor importancia, al no ser necesarios los valores originales de estos registros, pudiera ocurrir que aquéllos fueran imprescindibles para el correcto funcionamiento del programa. Este problema, como se puede ver en el programa EJEMPLO2.COM, se soluciona con el uso de las ya conocidas instrucciones PUSH y POP, que se emplean para introducir/sacar valores de dos bytes de la pila: los registros se salvan al comienzo de la subrutina y se recuperan al final en orden inverso. Como hay tantas instrucciones PUSH como POP, el registro SP no experimenta una variación neta.

Hasta el momento se ha visto la forma mínima de diseñar un procedimiento en ensamblador, y se ha estu-

diado el funcionamiento de las instrucciones implicadas en su llamada y retorno. Esto ya nos permite codificar programas empleando subrutinas, pero para alcanzar una verdadera flexibilidad en su manejo necesitamos estar en condiciones de cambiar el resultado de la ejecución de éstas en función de valores enviados desde el módulo principal. Es decir, debemos estudiar cómo pasar parámetros a una subrutina.

LA PILA COMO BUZON

También en este caso la comunicación entre el programa que llama y el procedimiento se hace a través de la pila: desde el primero se introducen en ésta los datos que va a emplear la subrutina en sus operaciones, y ésta, una vez en ejecución, accede a la pila para recuperarlos.

Los parámetros se pasan moviéndolos previamente y por turnos al registro AX y ejecutando a continuación la instrucción PUSH AX. Si el valor que se necesita pasar posee una longitud de un byte, se mueve al registro AL, el byte menos significativo de AX, y también se introduce en la pila mediante la instrucción PUSH AX. El hecho de que quede en aquélla un hueco de un byte no es importante, ya que lo que prima en este momento es la facilidad para recuperar los parámetros una vez en la subrutina, y es más sencillo sabiendo que éstos van de dos en dos.

Una vez la subrutina está en ejecución es necesario acceder a los parámetros. Ahora bien, ¿en qué posiciones se encuentran exactamente? Veamos: la subrutina "sabe" el orden en el que fueron pasados a la pila. Sabe también que cada uno de ellos ocupa dos bytes y, por último, que la instrucción CALL que la ha invocado ha introducido un valor adicional para preparar el camino de vuelta.

En el programa EJEMPLO3.COM se muestra, con el procedimiento VerCadena, un ejemplo de subrutina NEAR que trabaja a partir de un parámetro de dos bytes (el desplazamiento de comienzo de una cadena de caracteres) y de un parámetro de un byte (los colores que deseamos para la presentación de texto en pantalla). La figura 7 muestra cómo está la pila en el momento en

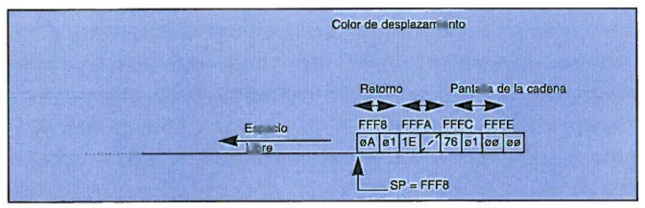


FIGURA 7.

que el control ha sido transferido a la primera instrucción de la subrutina.

El registro SP apunta a las posiciones en las que, en este caso, se almacena el desplazamiento de la instrucción de vuelta. Dos bytes más arriba está el último parámetro introducido (el atributo de pantalla), y otros dos bytes más arriba se tiene el primer parámetro que fue pasado. Como se puede apreciar, existe una regularidad que permite conocer en todo momento la posición relativa con respecto a SP (en el instante de comenzar la subrutina), de los parámetros pasados desde el módulo principal. Para hacerlos accesibles es necesario emplear un registro que permita acceder al contenido de las posiciones de memoria de la pila. Como ya sabemos después de haber visto los modos de direccionamiento, ese registro es BP: el registro puntero base.

Si en este instante de la ejecución, antes de salvar los registros empleados, SP se mueve a BP, el último parámetro introducido se puede referenciar como [BP+2], el penúltimo como [BP+4], etc, siendo posible ahora cargarlos en los registros oportunos de forma que la subrutina pueda realizar su labor.

La ejecución de una subrutina no debe modificar de forma neta la pila

¿Correcto todo? En principio parece que sí, pero ocurre lo siguiente: hemos supuesto que nuestro procedimiento es el único que debe acceder a la pila y no se ha salvado el valor antiguo de BP, por lo que si este bloque de instrucciones fue llamado desde otro que también maneja parámetros, se perderá por completo la situación de estos últimos. La catástrofe se cierne en el ambiente... Por supuesto este problema se obvia con el empleo de la instrucción PUSH BP al comienzo de la subrutina, pudiéndose mover ahora con toda tranquilidad SP a BP. Como la pila ha crecido en otros dos bytes, al último parámetro introducido se debe acceder como [BP+4], al penúltimo como [BP+6], etc. Esto se muestra en el programa EJEMPLO3.COM, donde el atributo de pantalla que emplea VerCadena se recupera de la pila mediante MOV BH, [BP+4] y el desplazamiento de la cadena que se visualiza, con la instrucción MOV DX, [BP+6]. La figura 8 ilustra el estado de la pila después de haberse ejecutado las dos primeras instrucciones del procedimiento.

Si la llamada hubiera sido FAR, la instrucción CALL habría almacenado, además del desplazamiento, el segmento de la instrucción siguiente, por lo que el espacio ocupado antes de la inicialización de BP sería dos bytes mayor. Así, el último parámetro introducido desde el programa principal sería accedido como [BP+6], el penúltimo [BP+8], y así sucesivamente.

REEQUILIBRANDO LA PILA

Además del almacenamiento e inicialización de BP, el manejo de parámetros tiene otras implicaciones a la hora de diseñar la subrutina. Pensemos si no en la finalización de este procedimiento: se recuperan los registros afectados (incluido BP) mediante la instrucción



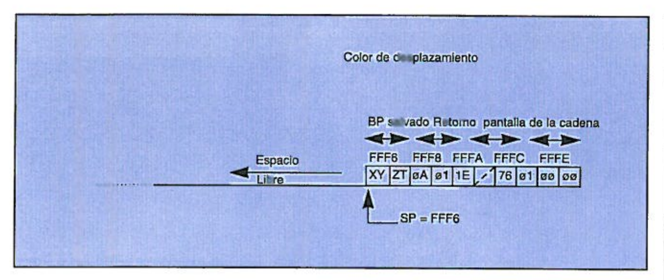


FIGURA 8.

POP, se ejecuta RET y el control se transfiere a la siguiente instrucción del programa principal. Fijémonos ahora en el valor de SP: es el que tenía antes de la instrucción CALL, pero NO es el que tenía antes de efectuarse el pase de parámetros. Por lo que respecta a la subrutina, ésta se comporta bien, pero dado que cada vez que sea invocada existirá un pase de parámetros, cada llamada implicará un aumento de la parte ocupada de la pila, y por ello se dispondrá cada vez de menos espacio en ésta.

Este problema se puede solucionar de varias formas, y la utilizada en el procedimiento VerCadena consiste en emplear una posibilidad de RET y RETF, que es el uso de un argumento que se suma al registro SP, una vez que han finalizado los pasos internos de estas instrucciones mostrados ya en las figuras 4 y 6. En el ejemplo, la instrucción RET 4 extrae de la pila el valor 010Ah, lo introduce en el registro IP, incrementa en dos unidades SP y, después, suma el valor 4 a este registro, de forma que se recupera el valor FFFEh que se tenía antes del pase de parámetros.

VARIABLES LOCALES

Hasta el momento se ha visto que cuando se trabaja con procedimientos, la pila se emplea de forma continua: para almacenar la dirección de retorno al módulo principal, para guardar temporalmente los registros empleados y para pasar los parámetros. Ahora veremos cómo extraer de ella un servicio más.

Dentro de una subrutina se pueden definir variables locales

Cuando se diseña el cuerpo de la subrutina, los protagonistas de las operaciones que tienen lugar dentro de ésta suelen ser los registros del procesador, una vez que los parámetros pertinentes han sido cargados en ellos. Si se repasa el código de los procedimientos de los programas ejemplo, se comprobará que esto ha sido así en todas las ocasiones.

Ahora bien, puede ocurrir que la subrutina sea demasiado larga o su código muy complejo y que, para efectuar operaciones con comodidad, no baste con esos registros y se necesiten más operandos. Si esto es así y se prevee en su diseño, el programador puede emplear la pila para crear variables locales dentro del procedimiento en cuestión mediante la simple operación de restar al registro SP el tamaño total deseado para estas variables. El momento idóneo para hacerlo es después de haber inicializado BP con el desplazamiento de la zona de parámetros de la subrutina y antes de salvar los registros que ésta emplea, debiéndose por este motivo acceder a las variables así construidas mediante expresiones de la forma [BP - nº]. El programa EJEMPLO4.ASM muestra mediante la subrutina Multiplica un ejemplo de procedimiento en el que se crea una variable local para facilitar los cálculos, y en la figura 9 se refleja la imagen de la pila que "ve" este procedimiento una vez en ejecución.

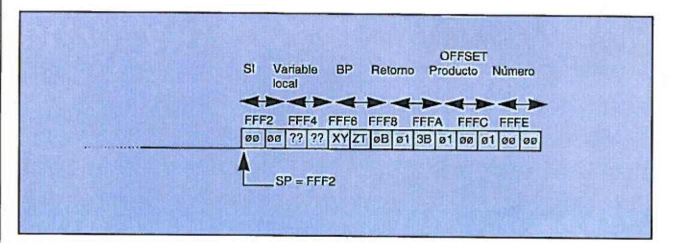


FIGURA 9.

Cuando la subrutina finaliza, el espacio asignado a variables locales debe ser eliminado, y para ello basta con sumar a SP el tamaño de las variables locales antes de ejecutar POP. Sin embargo, un método más elegante y rápido, empleado en el programa EJEMPLO4, es el de mover a SP el valor del registro BP en ese mismo momento: antes de ejecutar POP BP.

PARA FINALIZAR

A lo largo de estas páginas se ha visto cómo funciona la llamada y el retorno de un procedimiento, la forma de pasarle parámetros a través del segmento de pila, y cómo se pueden definir variables locales dentro de él. Esto nos permite codificar programas en ensamblador utilizando recursos totalmente equiparables en cuanto a su funcionalidad a los empleados en los lenguajes de alto nivel, aunque, eso sí, de una forma incómoda y engorrosa.

En el siguiente artículo se mostrará cómo se puede, con la ayuda de directivas del lenguaje, simplificar y automatizar al máximo la codificación y uso de una subrutina, de forma que ésta pueda manejarse con la misma soltura que un procedimiento o una función desde un lenguaje de alto nivel. ■

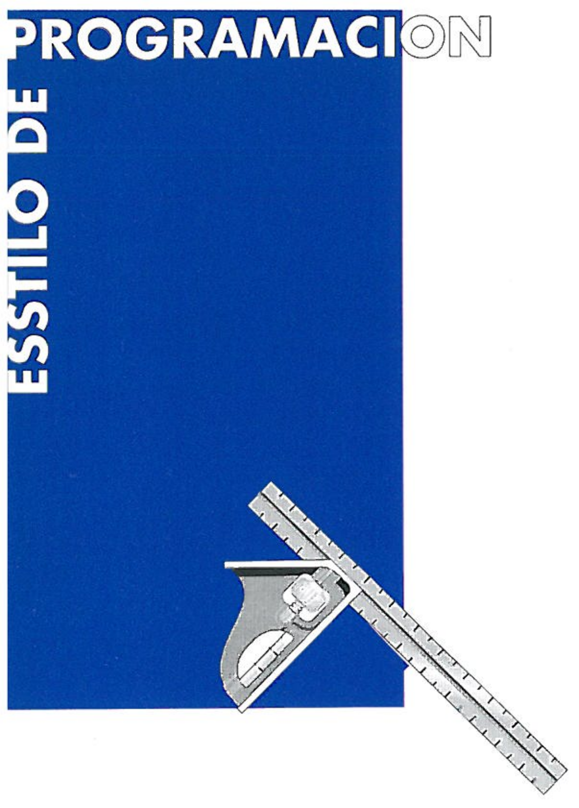
Bibliografía:

- *Assembler Inside & Out*, Harley Hahn, OSBORNE MCGRAW-HILL, 1992.

- *8088-8086/8087 Programación Ensamblador en entorno MS DOS*, Miguel Angel Rodríguez-Roselló, ANAYA , MADRID, 1988.

MODOS Y MANERAS

Fernando de la Villa



La legibilidad de un programa juega un papel muy importante en algunos aspectos del ejercicio de la programación. Uno de estos aspectos es el mantenimiento de software, esto es, la tarea de modificar un programa con el fin de corregir errores y añadir nuevas opciones. Está considerado como una de las etapas más costosas en el desarrollo de un programa, en términos de tiempo y dinero. Teniendo en cuenta que la mayor parte del tiempo que dedica un programador al mantenimiento se emplea en estudiar el código fuente del programa, se llega a la conclusión de que la claridad es vital para el adecuado desarrollo de una aplicación informática. Esto se hace patente de manera especial cuando el trabajo se realiza en grupo.

A un programador novel, acostumbrado a escribir programas pequeños, todo esto le puede parecer una exageración. En el momento de escribir el código, el autor sabe muy bien lo que hace el programa. ¿Para qué molestarse en ir formateando el texto y añadir comentarios?.

El mantenimiento de programas informáticos es muy costoso

El pequeño programa mostrado en el cuadro 1, funciona perfectamente. Pero es muy poco legible y la tarea que realiza está poco clara. Si el propio autor se ve en la necesidad de modificarlo varios meses después de haberlo escrito, se encontrará con que no se acuerda de cómo lo hizo y perderá un tiempo precioso en comprender su propio programa, para poder hacer los cambios necesarios. No es difícil imaginar lo que pasaría si el programa, en lugar de ser tan pequeño, estuviera compuesto por miles de líneas de código.

La versión del cuadro 2 es bastante más clara y legible que la anterior. Se ve rápidamente cuál es el objetivo, se conoce el significado de cada variable y de cada constante. Cualquier modificación que se realice ahora será mucho menos costosa, menos pesada y se tendrán más posibilidades de hacerlo de forma correcta.

El único inconveniente que se podría encontrar es la longitud que ha aumentado considerablemente.

Es incuestionable el hecho de que un programa se lee más veces de las que se escribe. Todo el esfuerzo que se realice en mejorar y facilitar esa lectura es beneficioso tanto para el autor, como para otros programadores.

Sin embargo, este hecho apenas afecta al código generado, y el gasto en espacio de almacenamiento que produce no es ni tan siquiera comparable a los beneficios que aporta.

NORMAS DE ESTILO

No se puede hablar de enumerar una serie de reglas fijas para la escritura de programas. Es un proceso creativo, y cada programador tiene sus métodos y sus manías. Sin embargo, se pueden establecer unas normas generales que ayudan a obtener programas más legibles.

La claridad es vital para el adecuado desarrollo de un programa

Los ejemplos que se van a citar a continuación están orientados al lenguaje C, pero son perfectamente aplicables a cualquier otro lenguaje de programación que sea estructurado y de formato libre.

IDENTIFICADORES

Un identificador es un nombre asociado a un objeto de programa (constantes, variables, funciones, procedimientos y tipos de datos) Como tal, debe identificar claramente al objeto del que se trata. El nombre elegido debe ser coherente con el significado del objeto, de tal manera que sea una descripción clara y breve del mismo.

Escoger un nombre adecuado no es tarea difícil: Un objeto de programa, al fin y al cabo, es la representación de una entidad del mundo real. Si una variable va a contener, por ejemplo, la estatura del cliente, lo lógico es utilizar un nombre como estatura_cliente. Los programadores inexpertos suelen emplear nombres cortos, fáciles de teclear, como "ec". Esto da como resultado programas crípticos, muy difíciles de entender. Pero esto no quiere decir que no se puedan utilizar nombres breves. Es práctica común utilizar nombres como "i", "j", "k", "l" para nombres de índices de bucle, por ejemplo, para calcular un sumatorio, ya que en la expresión matemática se utilizan índices como los citados. Por otro lado, no es conveniente elegir identificadores como "estatura_del_cliente_de_la_tienda". Esta sobrecarga de información en el nombre es totalmente innecesaria y origina lecturas muy pesadas.

A la hora de elegir un identificador para un procedimiento, se tendrá en cuenta solamente la misión que realiza, sin dejarse distraer por las operaciones necesarias para alcanzar su objetivo. El nombre más indicado para uno encargado de borrar la pantalla sería "borrar_pantalla". Es aconsejable utilizar el verbo en infinitivo, para destacar la acción concreta que lleva a cabo y para establecer una analogía directa entre el código y el análisis que se haya hecho del algoritmo. Nombres como "borra" o "b_pantalla" no son lo suficientemente significativos.

Las funciones devuelven un valor, por lo que su identificador debe expresar esto. Al hacerlo así, las expresiones en el programa cobran sentido por si solas:

```
acumulador = acumulador + factorial (contador);
```

CUADRO 1. Sentencias de control.

Sentencia IF:
```
if (condición) {
  sentencias
}
```
Sentencia IF-ELSE:
```
if (condición) {
  sentencias
}
else {
  sentencias
}
```

Sentencia SWITCH:
```
switch (expresión) {
  case expresión: sentencias
  break;
  case expresión: sentencias
  break;
  default: sentencias
}
```

Sentencia FOR:
```
for (exp_1; exp_2; exp_3) {
  sentencias
}
```

Sentencia WHILE:
```
while (condición) {
  sentencias
}
```

Sentencia DO-WHILE:
```
do {
  sentencias
} while (condición);
```

No existen unas reglas fijas para la escritura correcta de programas

CUADRO 2

```
char far*q=(char far*)0xB8000000;
void main(){
int i=2000;
while(i—){*q++=177;*q++=14;}
}
```

CONSTANTES

El uso de constantes en un programa facilita su comprensión, hace más fáciles los cambios y disminuye la posibilidad de cometer errores. Es un valor que permanece inalterado durante la ejecución del programa y puede ser usada una o varias veces a lo largo del mismo (Cuadro 4).

Otra ventaja que proporcionan es que si se necesita cambiar un valor constante, el cambio se hace sólo en la definición de la constante, en lugar de en todos y cada uno de los puntos del programa en los que aparezca el valor.

COMENTARIOS

La utilización de comentarios es un buen hábito que deben poseer todos los programadores. Aumentan la claridad de un programa, contribuyen de forma decisiva a la documentación y en muchas ocasiones ahorran quebraderos de cabeza. Se debe hacer buen uso, pero no abuso de ellos. Un programador que ponga comentarios a sentencias cuyo significado es evidente lo único que consigue es embrollar el código. Deben ser breves y concisos, evitando las divagaciones.

Se utilizan para explicar brevemente lo que hace el programa, las funciones y los procedimientos. También sirven para aclarar el significado de aquellas variables y constantes cuyo identificador no sea suficiente para entender su uso, y para informar sobre el objetivo de las sentencias más complicadas. Por último, se pueden añadir en los cierres de bloque para especificar a qué sentencias de control de flujo pertenecen dichos cierres, en los casos de fuerte anidamiento.

Se debe evitar en lo posible el uso de la sentencia GOTO

ESTRUCTURA DEL PROGRAMA

La forma en que se estructura el texto que compone un programa es un factor clave para que el programa sea legible y, por tanto, más fácil de mantener. Un programa tiene que ser como un buen libro: organizado, claro y fácil de leer y de entender.

Por suerte, casi todos los lenguajes de programación son de formato libre, por lo que podemos estructurar el código como más convenga. El programa se debe dividir en varias secciones: Presentación del programa, constantes, variables globales, funciones y procedimientos, y por último, el programa principal. Cada una de estas secciones debe estar separada de las demás por, al menos, una línea en blanco.

La presentación del programa es la sección en la que se da a conocer el programa. En forma de comentario se debe incluir el nombre del programa, una breve explicación de lo que hace, su autor, los parámetros de llamada si los tiene, y una lista con las modificaciones que ha sufrido acompañadas de la fecha en la que se hicieron. Por último se pueden añadir unas notas sobre el programa.

Un programa tiene que ser claro, fácil de leer y de entender

A continuación, se definirán todas las constantes, seguidas de la declaración y, si procede, definición de las variables globales del programa.

La sección de funciones y procedimientos requiere una especial atención. Cada módulo debe ser precedido, como comentario, por una descripción de la misión que desempeña, significado de los parámetros de llamada, y algún apunte que se crea necesario. En el caso

CUADRO 3

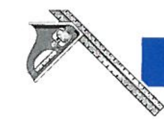
```
/*-------------------------------------------------------------
Programa que rellena la pantalla con un mismo carácter y
atributo, accediendo directamente a la memoria de vídeo. Se
asume tarjeta de vídeo no monocroma y modo texto 80x25.

Autor: Fernando de la Villa Albares
-------------------------------------------------------------*/

#define DIR_VIDEO 0xB8000000
#define CARACTER  177
#define ATRIBUTO   14
#define ANCHO      80
#define ALTO       25

/* Puntero para recorrer la memoria de video */
char far *MEM_VIDEO = (char far *) DIR_VIDEO;

void main ()
{
  int contador = ALTO*ANCHO;
  while (contador—) {
    *MEM_VIDEO++ = CARACTER;
    *MEM_VIDEO++ = ATRIBUTO;
  }
}
```



CUADRO 4

#define PI 3.1415927
*longitud_circulo = 2 * PI * radio*

Es mejor que:

*longitud_circulo = 2 * 3.1415927 * radio*

de las funciones, debe especificarse el significado del valor que se devuelve, por ejemplo, si puede retornar algún código de error. El módulo debe escribirse utilizando indentación o sangrado del texto, aumentando así la claridad del código, y evitando que el lector se pierda en la estructura debido al anidamiento. Algunos estudios demuestran que el indentado debe hacerse con 2, 3 ó 4 espacios en blanco. Utilizar uno grande (8 o más espacios) hace las líneas del programa demasiado largas en algunas ocasiones, por lo que resulta muy incómodo leerlas sobre una pantalla. Por esta misma razón, se deben evitar las líneas de código demasiado largas. También se debe intentar no escribir bloques de instrucciones muy grandes. Salvo raras excepciones, un bloque demasiado largo es producto de un análisis poco cuidadoso del módulo. Por tanto, será necesario dividir el bloque en varias llamadas a otros módulos, de tal modo que el código sea lo más fácil de seguir posible. Una cosa más: los módulos tienen que separarse por una o más líneas en blanco para distinguirlos fácilmente. También se puede añadir algo a la cabecera del módulo con el fin de destacar aún más la separación, como por ejemplo, una serie de caracteres iguales (guiones, asteriscos...)

No se puede descuidar el estilo a la hora de escribir comentarios

Los comentarios aumentan la claridad de un programa

El programa principal es simplemente un módulo más, por lo que se aplican las mismas normas que para cualquier otro módulo de programa, con la excepción de que no necesita una cabecera que explique su cometido por ser éste evidente.

LA SENTENCIA GOTO

Lo mejor que puede hacerse es huir del uso de esta sentencia. En un lenguaje estructurado no es necesaria, y contribuye enormemente a oscurecer el código. Como dicen Kernighan y Ritchie en su libro de C, su utilización sólo está justificada en casos muy especiales, como salir de una estructura profundamente anidada.

CONSIDERACIONES FINALES

La utilización de unas normas de estilo es tan importante, que todas las empresas serias dedicadas al desarrollo de programas imponen a sus empleados la rígida observación de determinadas normas. De esta manera se consigue una uniformidad en los resultados que facilita el intercambio de información entre los distintos departamentos, y posibilita la reestructuración del grupo de trabajo con el mínimo trastorno para el desarrollo de la aplicación.

A un programador acostumbrado a una forma de trabajo muy personal, le puede ser muy costoso el adaptarse a unas normas fijadas por la empresa. Esto es tan comprensible como habitual, por lo que se inventaron unas herramientas denominadas embellecedores de código. Estas utilidades cogen un programa fuente escrito con cualquier estilo de programación y devuelven el mismo programa, pero estructurado de una forma concreta. Así, un programador puede trabajar de la manera que le resulte más cómodo, y el trabajo finalizado que entrega se corresponde con las normas exigidas por la empresa. Sin embargo, estas herramientas no obran milagros, y el uso de comentarios y de identificadores adecuados corre siempre a cargo del programador.

Es aconsejable, además de seguir las reglas citadas anteriormente, el procurar escribir con estilo mecanográfico. Se deben utilizar espacios en blanco para separar expresiones y partes diferentes de una misma expresión. También de pueden usar paréntesis, aunque no sean necesarios, para clarificar una expresión complicada. El hacer muy compacta una línea de código no contribuye en absoluto a su legibilidad. Es mejor utilizar:

```
for (i = 0; i < MAX_ITERACIONES; i++) ...
```

Que la siguiente forma:

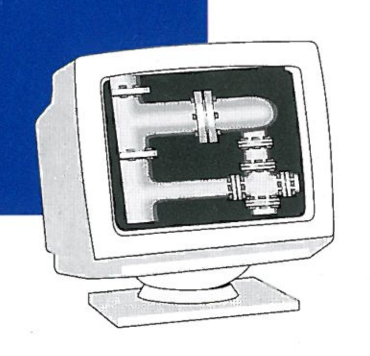
```
for(i=0;i<MAX_ITERACIONES;i++) ...
```

Tampoco se debe descuidar el estilo a la hora de escribir los comentarios. Son texto en castellano, y por tanto no hay que olvidar los acentos, ni los signos de puntuación. En las cabeceras de los módulos se pueden usar sangrados y utilizar líneas en blanco para organizar el texto. Un comentario mal escrito o mal expresado no sirve para nada, porque no cumple su cometido de informar.

En definitiva, un aspecto que podría parecer trivial en el proceso de creación de programas no lo es en absoluto. Los programadores inexpertos deben adoptar un estilo adecuado y los experimentados no deben ser perezosos y abandonar estas costumbres en programas pequeños. ■

TURBO PASCAL PARA WINDOWS

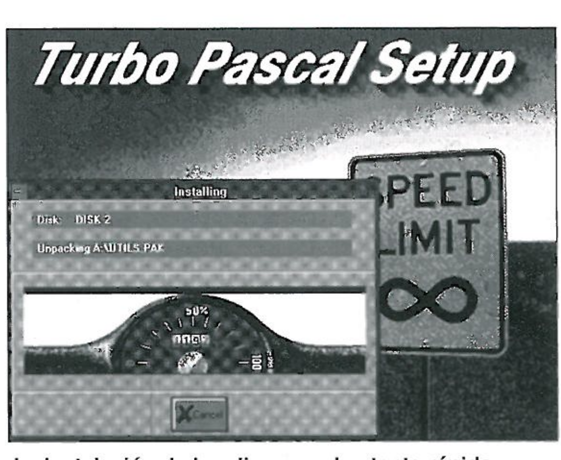
Cristina de la Fuente



En un principio, se diseñó este sencillo y práctico lenguaje como herramienta de ayuda a la enseñanza de la programación, todo ello bajo un entorno de programación realmente innovador por aquel entonces. Tras sacar al mercado nuevas versiones del lenguaje, ha logrado hacerse hueco en el mundo de la programación Windows. Hasta el momento, el lenguaje C era el único que permitía crear aplicaciones Windows, ahora con Turbo Pascal para Windows, se ha levantado la veda. Se trata de un nuevo producto, independiente del ya conocido compilador para DOS Turbo Pascal 6.0, que se ejecuta desde el entorno Windows y no desde el DOS. Un producto completo, en cuanto a funcionalidad, potente y amigable que hará las delicias de todos los aficionados a la programación Windows.

VISION DE CONJUNTO

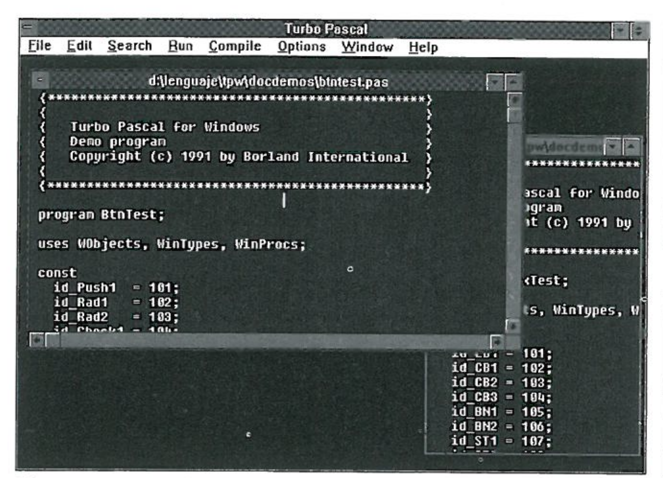
Hasta ahora la programación en Windows ha usado únicamente el lenguaje C y algunas herramientas de desarrollo que corren bajo DOS. Todo ello hacía que el proceso de desarrollo de programas fuera largo, complicado y confuso. Ahora han cambiado las cosas. Turbo Pascal para Windows ha logrado ponerse a la altura de sus competidores e incluso mejorado detalles tan importantes como el sistema de ayudas o el entorno de trabajo, una propia aplicación Windows. Además de un amigable juego de herramientas, cuya facilidad y operatividad están garantizadas, incorpora la implementación completa de las librerías API de Windows y ObjectWindows, así como un potente depurador, el Turbo Debugger para Windows, que trabaja en modo texto aún siendo una aplicación que corre bajo Windows.



La instalación de los discos es bastante rápida.

No se han olvidado detalles tan importantes como garantizar las características propias del Turbo Pascal: velocidad de compilación, unidades de código reutilizable y programación orientada a objetos, así como proporcionar el soporte para crear y utilizar librerías de enlace dinámico (DLL), inclusión de ficheros de re-

Puede decirse que gran parte del éxito de la compañía Borland se debe al trabajo realizado para que el lenguaje de programación Pascal se perfile prácticamente como la única alternativa al lenguaje C.



El entorno permite tener abiertas multiples ventanas de edición para facilitar la creación de nuevos programas.

cursos en ficheros ejecutables, nuevos tipos de datos y revisión de las unidades del Turbo Pascal para DOS.

Y cómo olvidarse de ofrecer al programador un entorno realmente completo y sencillo de manejar. Turbo Pascal para Windows es un programa Windows que se ejecuta bajo Windows y como tal debe garantizar un entorno integrado de desarrollo (IDE) que permita escribir, compilar y probar sus aplicaciones haciendo pleno uso de las capacidades multitarea de Windows.

Hasta ahora todo va bien. Pero ¿qué ocurre cuando necesitas desarrollar una aplicación que utilice recursos como iconos, cursores, cuadros de diálogo, bitmaps, etc?... se nos caería el mundo encima de no ser por el juego de herramientas Whitewater Resource Toolkit. Este juego de herramientas consiste en una serie de editores de recursos para facilitar la construcción de menús, cuadros de diálogo, teclas aceleradoras, bitmaps, iconos, cursores y cadenas de caracteres sin tener que programarlos. Así mismo, dispone de un Gestor de recursos que sirve, como su nombre indica, para realizar todas las gestiones pertinentes de copiado, borrado, renombrado e inclusión en ficheros existentes de todo tipo de recursos.

Una simple directiva de compilación se encarga de llamar al compilador de recursos para que efectúe su trabajo desde el propio entorno integrado de desarrollo. Puede, igualmente, compilar recursos escritos por otros programadores de Windows.

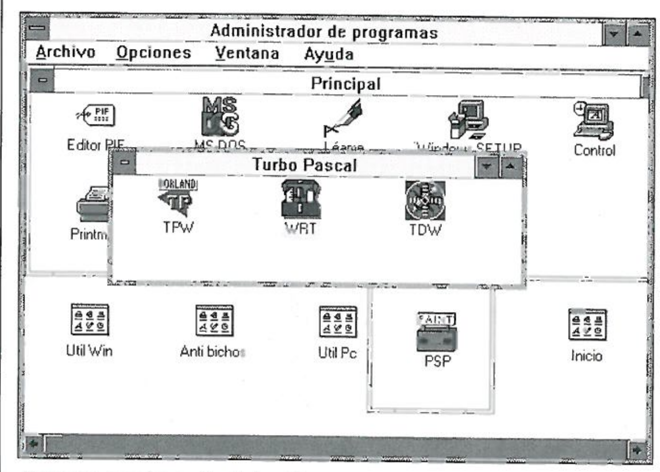
Posiblemente la parte más frustrante a la hora de enfrentarse al desarrollo de una aplicación Windows es determinar qué acciones necesita contemplar para que el programa funcione en Windows y además asegurarte que haces todas.

Además una aplicación Windows debe saber cómo responder al torrente de mensajes que Windows envía a sus aplicaciones en respuesta a sucesos de usuario tales como seleccionar una opción de menú. Sin embargo, Turbo Pascal para Windows no te deja todo el trabajo para ti solo. Te proporciona la librería orientada a objetos ObjectWindows que encapsula la mayoría de los comportamientos de una ventana y permite heredarlos en lugar de tener que reinventarlo todo cada vez que comienzas un nuevo programa. Proporcionándote este estable y sólido sistema, serás capaz de centrarte en las partes del programa que quieres escribir, en lugar de las partes que son comunes a toda aplicación Windows.

Una vez resueltos todos estos problemas, toma especial importancia el Turbo Debugger para Windows. Un potente depurador de alto nivel que te ofrece un completo soporte para evaluar y ensamblar expresiones, y depurar aplicaciones Windows y programas orientados a objeto. Al ser Turbo Pascal para Windows una aplicación Windows puedes llamar al depurador desde el propio entorno de desarrollo.

ENTORNO DE DESARROLLO INTEGRADO

Antes de comenzar la instalación de Turbo Pascal para Windows, hay que asegurarse de que el disco duro cuenta al menos con 6,5 Mb de espacio libre disponible. Para su correcto funcionamiento se precisa un PC 286 o superior, con al menos 1 Mb de memoria RAM y tarjeta gráfica VGA o superior. También es necesario disponer de ratón, ya que parte de las funciones del producto son inaccesibles sin él.



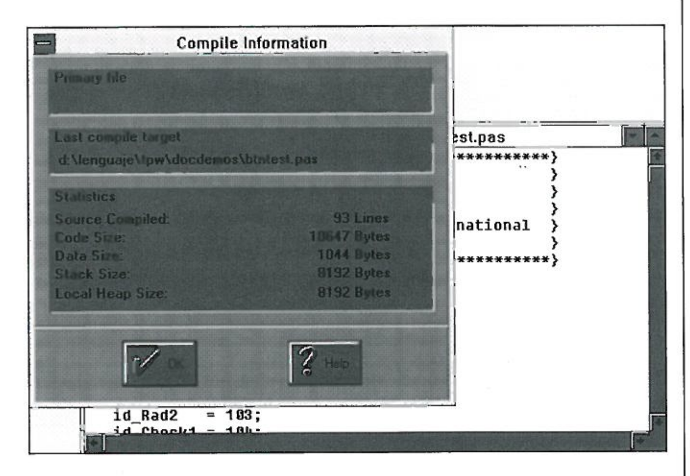
Cuando acaba la instalación, se crea un grupo de programa específico dentro del escritorio de Microsoft Windows.

Mientras que en la mayor parte de los programas la instalación suele causar algunos pequeños problemas, en Turbo Pascal para Windows es todo lo contrario. Desde un principio se muestra una pantalla cargada desde Windows. No tienes que preocuparte de nada, ya que todo se realiza de manera automática. Al finalizar, nos encontramos dispuestos en un grupo 3 iconos correspondientes al generador de recursos Whitewater Resource Toolkit (WRT), al Turbo Depurador (TDW) y al Turbo Pascal (TPW).

Arrancamos el programa Turbo Pascal para Windows pulsando dos veces consecutivas con el ratón sobre el icono TPW. Inmediatamente aparece una ventana de trabajo. En la parte superior se dispone de 2 barras. Por una parte está la típica barra con el título

Turbo Pascal, los botones de maximización y minimización y el botón del menú de control; y debajo se encuentra la barra de menús habitual de Windows (File, Edit, Search, Run, Compile, Options, Window y Help). En la parte inferior de la ventana está la barra de estado, que comenta brevemente la acción que estamos realizando, los fallos en la compilación, la línea y columna donde aparece el cursor en cada momento, etc. Y ya, en el centro de la ventana, se dispone del área de trabajo donde se crearán, abrirán y ejecutarán las aplicaciones, siempre dentro de ventanas del tipo Windows.

Turbo Pascal para Windows es capaz de gestionar múltiples aplicaciones en uno o más rectángulos llamados ventanas. Estas ventanas pueden moverse, cambiar de tamaño, y temporalmente esconderse como iconos permitiendo al usuario cambiar de tarea rápidamente. Esto es lo que se llama interfaz de múltiples documentos (MDI). El interfaz de múltiples documentos es un conjunto de convenciones de interfaz de usuario para crear ventanas que contienen ventanas hijas dentro de ellas. El entorno de desarrollo integrado de Turbo Pascal (IDE) es un ejemplo de MDI. El usuario puede abrir varias ventanas de edición en la mesa de trabajo de Turbo Pascal.



Una vez compilado un programa se puede obtener información detallada del resultado del proceso de traducción.

LA PROGRAMACION EN WINDOWS CON TURBO PASCAL PARA WINDOWS

Al ser Windows un entorno multitarea es necesario poder atender a diversidad de sucesos (acciones del hardware), que pueden provenir de diferentes aplicaciones o instancias. La programación basada en sucesos fuerza un cambio radical en el modo de plantear las estrategias de diseño de código secuencial. También se pueden simular sucesos forzando acciones por medio del software en ejecución.

Esta nueva idea de programación de aplicaciones consiste básicamente en manejar todas las entradas de usuario como sucesos. Éstos a su vez se transforman en mensajes y son despachados ordenadamente por la aplicación involucrada. Esta arquitectura conducida por sucesos se corresponde bastante bien con la aproximación orientada a objetos del Turbo Pascal.

Turbo Pascal for Windows facilita la programación en este entorno gráfico

Windows proporciona una facilidad para mantener las descripciones de recursos fuera del código fuente de una aplicación. Los recursos de una aplicación son unidos a su fichero ejecutable antes de ejecutar la aplicación. El separar la especificación de recursos del código fuente tiene un beneficio añadido: el aspecto de una aplicación puede cambiarse sin afectar al código fuente del programa; incluso no necesitas tener el código fuente para modificar los recursos de una aplicación. No hay que olvidar que Turbo Pascal para Windows incluye un conjunto completo de editores para crear recursos.

Así mismo, se pueden escribir módulos de librería para que las aplicaciones puedan cargarlas y liberarlas en tiempo de ejecución. Estos módulos deben estar en un formato ejecutable especial llamado librería de enlace dinámico (DLL).

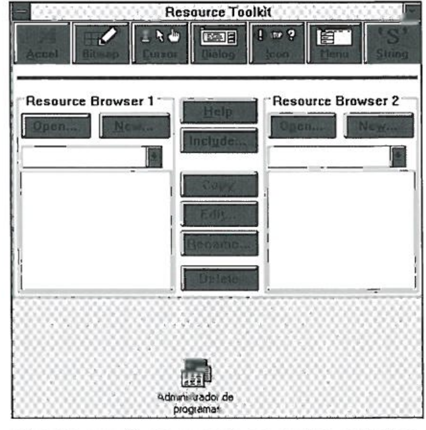
Otra característica es el intercambio dinámico de datos que no es más que otro protocolo de transferencia de información. Una aplicación transfiere información a otra mediante el envío de mensajes DDE.

Turbo Pascal para Windows unifica el proceso de escritura en pantalla e impresora en un único módulo llamado interfaz gráfico, el cual proporciona un interfaz común para todo programa Windows.

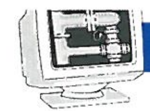
LA LIBRERIA ESPECIFICA PARA EL ENTORNO WINDOWS: OBJECTWINDOWS

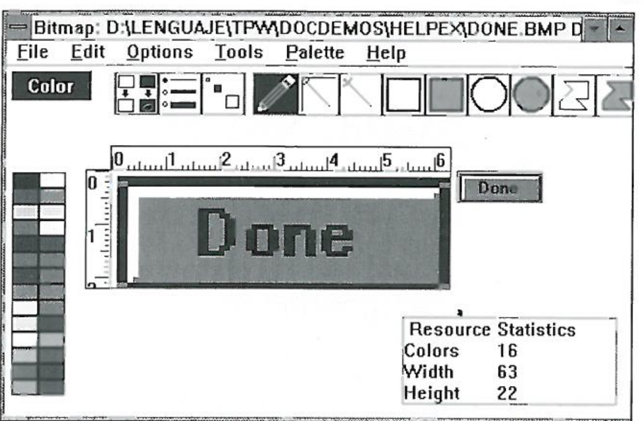
La programación orientada a objetos simplifica la tarea de programación para un entorno de representación de ventanas, permitiendo al desarrollador de la aplicación centrarse en la función de la aplicación antes que en su forma.

La ventana ObjectWindows y los tipos de objeto de la aplicación, como cajas de diálogo y controles, gestionan el comportamiento del tratamiento de mensajes requerido de un programa Windows, simplificando sumamente la



Desde aquí se puede acceder a todas las funciones del W. Resource Toolkit.





Se pueden crear botones personalizados para usarlos en las aplicaciones, añadiéndolas una imagen más original.

interacción del programador con el usuario.

Como resultado, se puede utilizar Turbo Pascal para escribir programas Windows con mucho menos tiempo y esfuerzo que con aproximaciones no orientadas a objetos.

ObjectWindows proporciona 3 características útiles:
- encapsulación de información Windows,
- abstracción de funciones API de Windows,
- respuesta automática a los mensajes.

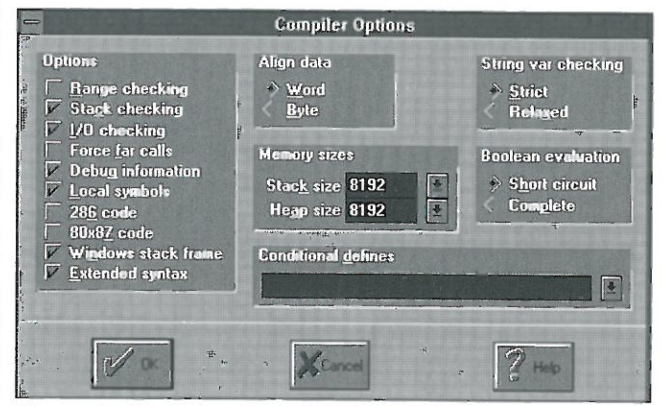
Siempre que se ejecuta una aplicación Windows aparece en pantalla una ventana donde se van a realizar todas las tareas necesarias. Esta ventana es, a su vez, la encargada de recibir todo tipo de sucesos Windows en

EJEMPLO DE UNA APLICACION UTILIZANDO OBJECTWINDOWS

```
Program MiPrograma;
uses WObjects;
type
TMiAplicacion = object(TApplication)
procedure InitMainWindow; virtual;
end;
procedure TMiAplicacion.InitMainWindow;
begin
MainWindow := New(PWindow, Init(nil, 'Construcción
de una ventana con ObjectWindows'));
end;
{        Programa principal        }
var
MiAplicacion: TMiAplicacion;
begin
MiAplicacion.Init('MiPrograma');
MiAplicacion.Run;
MiAplicacion.Done;
end.
```

forma de mensajes y procesarlos debidamente. Si tuviésemos que programar todo este cúmulo de sucesos, que pueden ocurrir en cualquier momento de la ejecución, necesitaríamos muchas líneas de código y no olvidarnos del más mínimo detalle para que todo funcione correctamente.

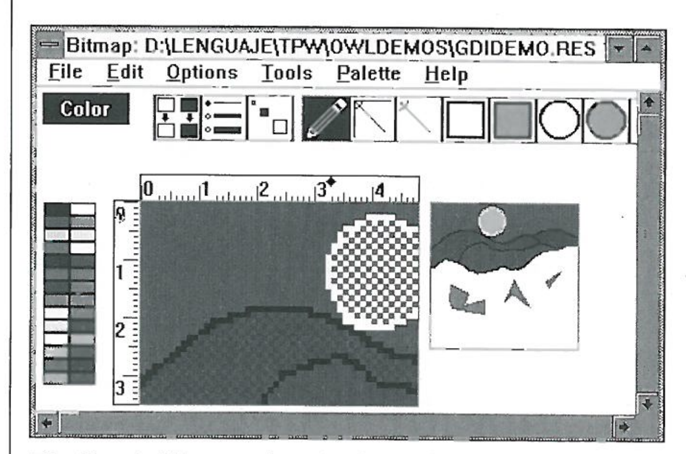
Gracias a la librería de objetos ObjectWindows, que encapsula una serie de comportamientos comunes, podemos abstraernos de toda la complejidad que aporta la programación Windows simplificando en gran medida el trabajo. Así, ObjectWindows proporciona objetos ventana, objetos diálogo, objetos de control, etc que se en-



El compilador se puede configurar a través de esta ventana.

cargan de resolver por nosotros todos los comportamientos predeterminados.

Así, en el programa de ejemplo se muestra una ventana vacía con el título Construcción de una ventana con ObjectWindows, con los botones de maximización, minimización y el botón de menú de control. No desempeña ninguna acción en sí, sólamente se espera de la ventana que actúe como tal, es decir, que se pueda mover, maximizar, minimizar, cambiar de tamaño y cerrar. Para ello es necesario incluir la librería



El editor de Bitmaps ofrece las herramientas para dibujar las pequeñas ilustraciones requeridas en los programas.

ObjectWindows con la sentencia uses WObjects. Todo programa ObjectWindows debe definir un nuevo tipo de aplicación, en nuestro ejemplo TMiAplicacion, derivado del tipo TApplication. Hay que inicializar, procesar y finalizar la aplicación con los métodos Init, Run y Done respectivamente. El constructor Init se encarga de cre-

ar una nueva aplicación dándole un nuevo nombre y crea y muestra la ventana llamando, entre otros, al procedimiento InitMainWindow. El método Run es el que pone en funcionamiento el bucle de recogida y proceso de mensajes hacia la nueva aplicación. Mientras que el destructor Done comprueba si todo está preparado para cerrar la aplicación y de ser así la cierra.

EL INTERFAZ DE DISPOSITIVOS GRAFICOS

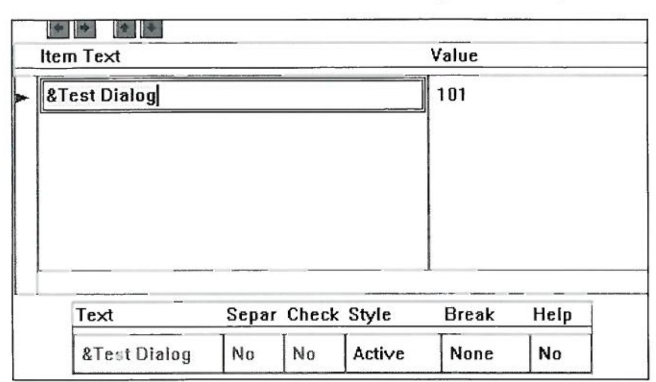
Muchos tipos de aplicaciones Windows necesitan solamente ventanas, cajas de dialogo y controles para un completo interfaz de usuario. Sin embargo, ciertas aplicaciones que dibujan y manipulan imágenes requieren gráficos para rellenar sus ventanas. Estos gráficos pueden ser en forma de líneas, figuras, texto y bitmaps.

Para dotar a las aplicaciones de funcionalidad gráfica, Windows tiene un conjunto de funciones llamadas interfaz gráfico (GDI).

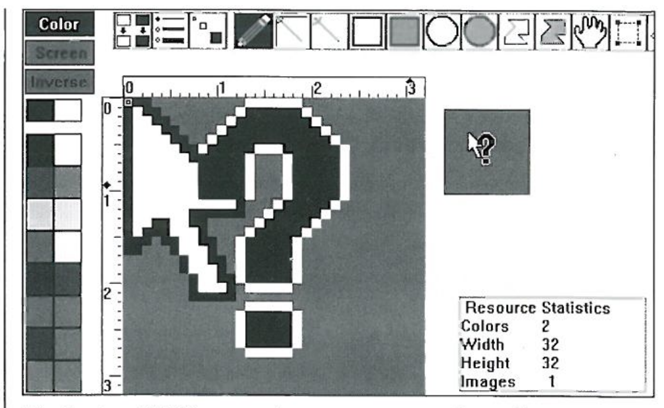
Para realizar la mayor parte de los dibujos se utilizan tres herramientas: un lápiz, un pincel y una herramienta para el tipo de letra. El lápiz se usa para dibujar líneas, arcos y segmentos. El pincel se usa cuando hay que rellenar figuras cerradas como rectángulos, rectángulos redondeados, elipses y polígonos. La herramienta de texto se usa cuando dibujas texto, especificando la altura, anchura, estilo (cursiva, negrilla, subrayado...), familia (Courier, Roman, Modern, System, Swiss, Terminal...),etc.

EL JUEGO DE HERRAMIENTAS WHITEWATER RESOURCE TOOLKIT

El juego de herramientas para la construcción de recursos Whitewater Resource Toolkit proporciona las herramientas necesarias para construir y modificar los recursos que se van a utilizar en la aplicación. Estas herramientas incluyen editores para las teclas aceleradoras, los mapas de bits, los cursores, los iconos, las cajas de dialogo, los menús y las cadenas de caracteres. Con ellos se pueden crear y editar recursos. Además se incluye una herramienta muy útil llamada Gestor de recursos. A partir de ella se puede acceder directamente a cada uno de los editores de recursos y además permite



Una vez creado los iconos se les asigna un nombre en este editor.



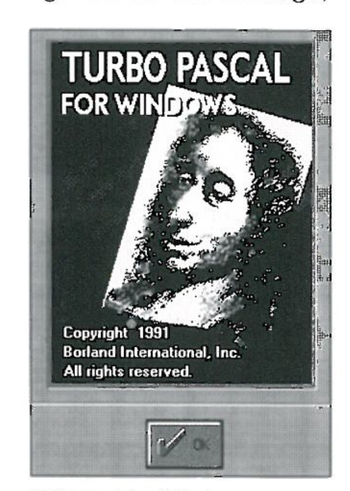
Mediante el WRT se pueden crear nuevos tipos de cursores.

efectuar diversas tareas como copiar recursos de un fichero a otro, borrar recursos de un fichero, renombrar los recursos, etc.

En la mayoría de los lenguajes, incluido el Turbo Pascal para Windows puedes crear iconos, cajas de diálogo, menús y, en general, todos los recursos que necesites directamente en el código fuente. Sin embargo, definiéndolos como recursos independientes del código fuente, eliminas las responsabilidades de gestión del código.

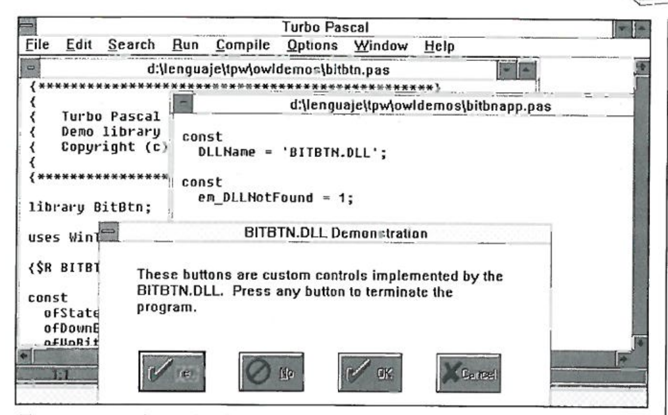
CONCLUSION

No hay duda de que nos encontramos con un producto realmente completo y potente que, desde un comodo entorno integrado de trabajo, nos proporciona todo tipo de facilidades. Turbo Pascal para Windows, al ser el mismo una aplicación Windows, aporta desde un principio familiaridad con los comandos del menú. Hemos comprobado como incluye una serie de librerías fundamentales, como ObjectWindows y las librerías API de Windows, para la construcción de aplicaciones completas y potentes, así como las herramientas de programación: Turbo Debugger para Windows, el generador de recursos y los compiladores de ayuda y de recursos... ¿Qué más se puede pedir?.



T.Pascal for Windows es una adaptación del T.Pascal v6.

Turbo Pascal para Windows está pensado para proporcionar al programador todo tipo de facilidades, y no podía olvidarse de una característica tan importante como es el sistema de ayudas. Posee un sistema de ayuda hipertexto además de un conjunto de manuales que complementan toda la información necesaria para que el trabajo sea lo más sencillo posible. Realmente no se puede decir que no ofrezca variedad de prestaciones. Turbo Pascal para Windows acaba de comenzar a andar y puede, por tanto, mejorar. Para todos aquellos a los


Este es un ejemplo de una aplicación oracle con TPW.

que les apasione la programación en Windows y les guste el lenguaje Pascal este es un producto que no deben dejar pasar.

GLOSARIO

– *Acelerador*

Un acelerador consiste en una tecla o combinación de ellas que pulsas para elegir una opción del menú más rápidamente.

– *Bitmap*

Constituyen un grupo de datos que la aplicación utiliza para mostrar una imagen en pantalla.

– *Botones de control*

Se encuentran formando parte de la estructura de un cuadro de diálogo o ventana y sirven para desencadenar una acción inmediata al ser seleccionados.

– *Cadena de caracteres*

Texto que la aplicación muestra en sus menús, cajas de diálogo, mensajes de error, mensajes del sistema, mensajes de estado, etc o que capta de una ventana.

Los sucesos suelen ser movimientos del ratón o, acciones del teclado

– *Cuadro de diálogo*

Ventana especial que contiene controles tales como cajas de listas, botones de control y campos de edición.

– *Control*

Elementos gráficos que componen el interfaz de usuario del diálogo.

– *Cursor*

Puntero para seleccionar elementos gráficos de una pantalla o para localizar la posición de inserción para la entrada.

– *Editores de recursos*

Editores capaces de editar recursos como teclas aceleradoras, menús, diálogos, mapas de bits, iconos, cursores y cadenas de caracteres sin tener que programarlos.

– *Enlace dinámico*

Proceso de conexión de diferentes componentes de un programa. El enlace dinámico ocurre mientras el programa se está ejecutando.

– *Entorno integrado de desarrollo (IDE)*

Entorno de desarrollo que permite escribir, compilar y probar las aplicaciones Turbo Pascal mientras se hace pleno uso de las capacidades multitarea de Windows.

– *Gráfico*

Elemento que representa tanto gráficos (mapas de bits, rectángulos, ...) como texto .

– *Icono*

Es otro tipo de mapa de bit especializado que representa una aplicación o una acción dentro de la aplicación.

– *Menú*

Un menú lista una serie de opciones de programa disponibles que se pueden activar. Cuando se elige una opción de la lista, puede suceder que o bien se abra otro menú para proporcionar opciones adicionales o bien se lleve a cabo una acción.

– *Mensajes*

Los mensajes son paquetes de información, que codifican lo acontecido acerca de un suceso.

– *ObjectWindows*

Es una librería orientada a objetos que encapsula la mayoría de los comportamientos que cada ventana tiene que conocer y permite heredarlo todo en lugar de tener que reinventarlo todo cada vez que comienza un nuevo programa.

– *Suceso*

Los sucesos suelen ser movimientos del ratón, acciones del teclado, alarmas del temporizador y, en general, todas aquellas acciones que puedan reclamar asíncronamente la atención del sistema.

– *Ventana*

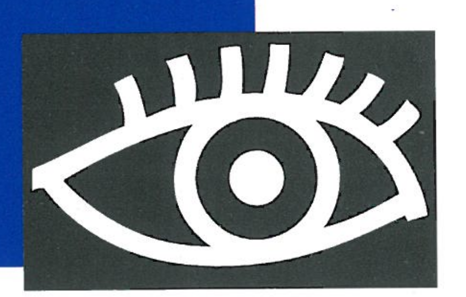
Objeto rectangular sobre la pantalla. Una ventana recibe entradas del usuario a través del teclado o del ratón y representa salidas en modo gráfico sobre su superficie. Contiene, generalmente, una barra de título identificando al nombre del programa, una barra de menú, un botón para el menú de control, botones de maximización y minimización, y barras de desplazamiento. ■

FICHA TECNICA

Nombre: Turbo Pascal for Windows 1.5.
Fabricante: Borland.
Distribuidor: Borland. Edf. Borland, (Miniparc 1) Azalea, 1. Soto de la Moraleja. 28100 Madrid.
Requisitos: 386 sx, 20 Mb libres en disco duro, 4 Mb RAM, Microsoft Windows 3.1.
PVP: 12.900 (+ IVA).

INICIACION A VISUAL BASIC 3.0

Sonia Sancibrián y
Fernando Cardo



Este artículo sobre VISUAL BASIC 3.0 pretende ser como su título indica una iniciación a éste, ya que para aprender cualquier lenguaje de programación hay que "pelearse" bastante con él.

VISUAL BASIC 3.0 es un interprete/compilador para Windows pero con grandes diferencias con respecto al BASIC standard. Aunque el código es compatible con los intérpretes y compiladores anteriores de Microsoft, la forma de trabajo es totalmente distinta. A la hora de desarrollar una aplicación VISUAL BASIC 3.0 para Windows, el diseño del interfaz es una de las principales tareas, y el código es muy dependiente de éste.

Es un BASIC parcialmente orientado a objetos, ya que no cumple una de las tres características de este tipo de lenguajes: encapsulación, polimorfismo y herencia.

Con respecto a la encapsulación se puede decir que la utiliza a la perfección aunque de manera diferente a C++. Los objetos de VISUAL BASIC 3.0 tienen unas propiedades como color, tamaño, tipo de letra y además incorporan unos métodos que responden a mensajes como clicks de ratón, pulsación de teclas, etc. en los que se puede integrar código. El polimorfismo es la capacidad de reaccionar de distinta manera ante un mensaje idéntico dependiendo del objeto que lo reciba, esta característica también es soportada, por ejemplo un click de ratón hará cosas distintas si lo recibe un botón o una barra de título de una ventana. Por último queda la herencia. No está soportada de ninguna manera, nada puede derivarse de otro objeto.

El lenguaje ha sido adoptado ya por muchas empresas de programación gracias a que se consiguen tiempos de desarrollo espectacularmente bajos y por la facilidad de ampliación que supone el poder cargar en los proyectos librerías con nuevos controles para el interface ".VBX" (VISUAL BASIC EXTENDED) y llamar a funciones de librerias ".DLL" (DYNAMIC LINK LIBRARIES)simplemente declarándolas.

Si usted ya ha programado en BASIC encontrará sencilla la adaptación a la nueva forma de diseño, ya que es el mismo código distribuido de distinta forma y en lugares "estratégicos". Las aplicaciones desarrolladas en VISUAL BASIC 3.0 no son como las clásicas aplicaciones de diseño "Top-Down", en las que el código estaba "apiñado" en un procedimiento principal que ocasionalmente, llamaba a procedimientos y funciones , sino que son aplicaciones orientadas a eventos, es decir se codifica en pequeñas porciones de código distribuidas por los controles visuales que com-

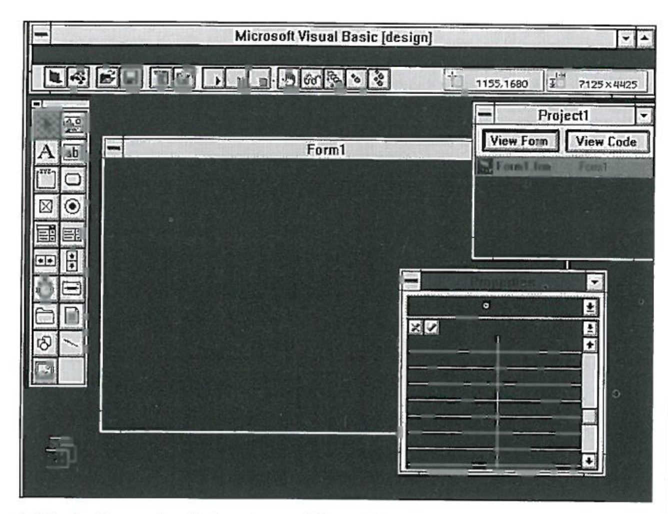


FIG. 1. Aspecto del entorno Visual Basic 3.0

ponen el interfaz (se podría decir que cada porción es un programa). Si ha seguido el hilo de lo explicado se preguntará: Y si no hay un procedimiento principal, ¿por donde empieza el programa?. La respuesta es sencilla: el sistema (Windows) carga el "form" (ventana de interface) que se ha declarado como principal y espera a que se interactúe con alguno de los controles que hay sobre ella, cuando esto ocurra, el código escrito para esta interacción tomará el control del programa, y cuando acabe volverá al estado de espera típico de las aplicaciones Windows, durante el cual se podrá ir a otro programa si se desea.

Pero empecemos aclarando una serie de conceptos que nos resultarán muy útiles a la hora de diseñar una aplicación: Un *evento* es un aviso al programador de que ha llegado un mensaje a un control, como puede ser un doble click de ratón, para que pueda programar la reacción. La diferencia entre mensaje y evento ha de quedar muy clara, no todos los mensajes provocan eventos. Por ejemplo una etiqueta obviará todos los mensajes que reciba del teclado, ya que no es un objeto diseñado para este propósito. Sin embargo un botón aceptará el mensaje "click" procedente del ratón y después de hundirse y levantarse, generará el evento "click" donde el programador habrá puesto su código. Una forma de saber los eventos que un control provoca al recibir mensajes, es ver en la ventana de código todos los procedimientos en los que se puede introducir código para ese control. Otro concepto es propiedad (property), una característica de un objeto VISUAL BASIC 3.0, como puede ser su tamaño, color o contenido.

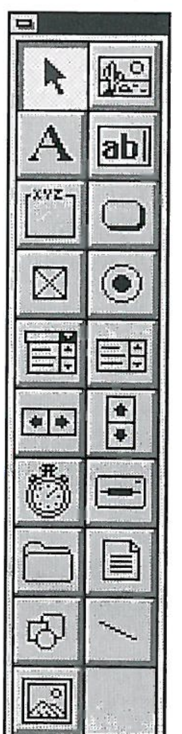


FIG. 3. Detalle de la barra de herramientas.

Un proyecto es un conjunto de ficheros necesarios para componer una aplicación

Para la correcta utilización de este texto y ejemplos, se ha supuesto que Ud. conoce el entorno Windows y está familiarizado con el uso del ratón y las ventanas, que sabe crear directorios desde Windows o MSDOS y grabar y copiar ficheros desde o a un disquete. Las expresiones que se utilizan para definir las partes y controles de VISUAL BASIC 3.0 se dan en Inglés. Hay que acostumbrarse a los términos anglosajones ya que la mayoría de los controles (por no decir todos) así como el propio VISUAL BASIC 3.0 están desarrollados en Inglés. Siempre es mejor hacer un pequeño esfuerzo al principio, que más adelante dará sus frutos en forma de rapidez y comprensión.

También es muy importante contar con una licencia de VISUAL BASIC 3.0 ya que el artículo está diseñado sobre esta versión y los ejemplos no se cargarán en otra anterior.

EL INTERFAZ PARA EL PROGRAMADOR DE VISUAL BASIC 3.0

Como en todos los programas nuevos, lo primero que hay que hacer es conocer un poco el interfaz con el que se trabajará. Conseguir rapidez y aprovechar al máximo las funcionalidades que ofrece, es básico para el tiempo de desarrollo de los proyectos posteriores.

Al arrancar, VISUAL BASIC 3.0 ofrece un proyecto nuevo que consta de una pantalla de programa *Form1,* (figura 1) donde se podrán añadir botones, cajas de texto y otros controles que encontraremos en la toolbox de controles (figura 3), si abrimos la ventana de

El programa permite efectuar rápidos cambios directamente en memoria sin tener que recompilar

proyecto (figura. 4) observaremos que sólo hay un fichero que es "form1.frm form1" el primer nombre indica cómo se llamará en el directorio de DOS donde se grabe y el segundo indica cómo se referenciará en nuestro programa. Si abrimos la ventana de propiedades (figura 5) desde el menú o la botonera, se desplegará mostrando las propiedades del control seleccionado en ese momento, en este caso el form. Desde aquí se pueden cambiar muchas de ellas como el nombre con el que se hará referencia a este form en el programa. Si se hace un doble click encima del form aparecerá una ventana con la cabecera de un

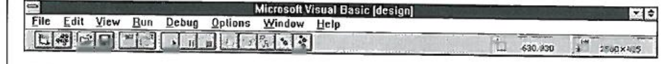


FIG 2. En el menú principal están todas las opciones disponibles.

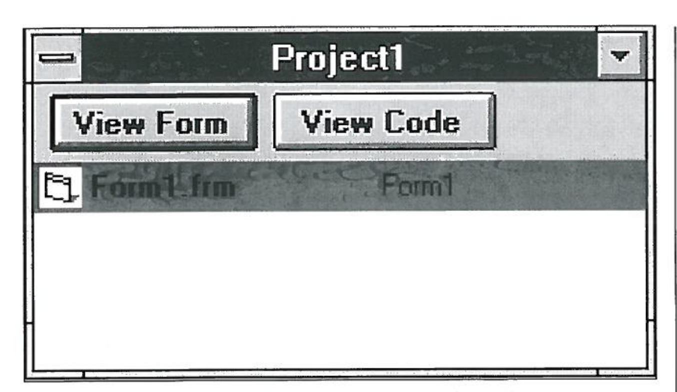


FIG 4. Ventana donde se ve el fichero que contiene el programa.

procedimiento donde aparece la palabra "load", este es uno de los procedimientos donde se reciben los mensajes que manda el sistema o el usuario, en este caso se llama durante el tiempo de carga del form. En esta misma pantalla se observan dos listas desplegables: la de la izquierda contiene todos los controles que pertenecen a ese form y la de la derecha los distintos eventos que genera al recibir mensajes, el control seleccionado en el de la izquierda. Si usted ya es programador se preguntará si es necesario repetir el mismo código en dos eventos que hagan lo mismo. Pues no, se pueden crear "Sub's" (procedimientos) o "Function's" (funciones) que serán llamadas desde estos.

Ahora que ya se conoce (más o menos) dónde está cada cosa podemos pasar a crear la primera aplicación.

PRIMERA APLICACION

Para diseñar cualquier aplicación debemos seguir tres pasos perfectamente definidos que nos ayudarán a diseñar la aplicación. Estos tres pasos son: diseño del interface, asignación de propiedades y adición de código a los eventos de los controles.

Para el diseño del interface utilizaremos la toolbox y la ventana *form*

Esta primera aplicación consta de una ventana (form) con dos controles: un "text box" (caja de texto) y un "command button" (botón de órdenes). El ejemplo es una sencilla aplicación que responde al pulsar el botón poniendo un texto que se conoce previamente en el text box. Con este sencillo ejemplo se puede ver claramente como se responde a un suceso (mensaje) generado por el usuario y cómo usar la propiedad "text" de un text box, y sobre todo, la parte básica del diseño de interface y cambio de propiedades en tiempo de diseño.

Aunque los controles que se ponen sobre una ventana pertenecen a esta, la ventana no deja de ser otro control con sus propiedades y eventos, aunque con la posibilidad de englobar otros controles. Se suele decir que el form es el padre de los controles y los controles son hijos del form. Esto se ve claro cuando se trata de programas MDI (Multiple Document Interface) en los que una ventana es la principal y tiene como "hijas" a otras ventanas que se pueden minimizar u ocupar toda la pantalla.

DISEÑO DE INTERFACE

Antes de empezar a desarrollar el interface merece la pena echar un vistazo al menú "options" en "environment" (figura 2). Esta opción de menú permite cambiar el aspecto de VISUAL BASIC 3.0 a nuestro gusto, aunque por ahora las opciones que más interesan son las de "Grid" (cuadrícula), ya que son las que afectan directamente al diseño de interface. Las opciones "Grid height" (alto de la cuadrícula) y "Grid width" (ancho de la cuadrícula) nos permiten definir una cuadrícula sobre el "form" (ventana sobre la que se diseña el interface o parte de él), para que se vea la cuadrícula habrá que poner a "True" (Verdadero) la opción "Show Grid" (mostrar la cuadrícula). Con esto se conseguirá una guía visual para la correcta alineación de los controles visuales en la ventana. Si aún así el pulso no es lo suficientemente firme como para colocar los controles alineados, se ofrece la opción "aling to grid" (alinear a la cuadrícula), que lo que hace es obligar a los controles que se añaden a "moverse a saltos" según el ancho de la cuadrícula.

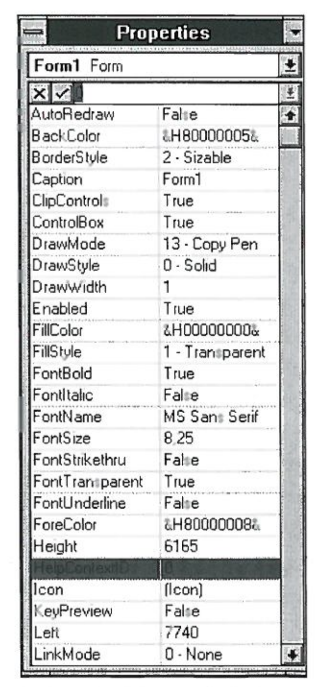
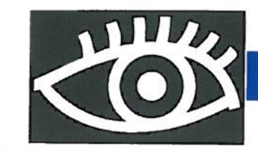


FIG. 1. Tabla propiedades.

Para el diseño del interface utilizaremos la *toolbox* y la ventana *form*, donde colocaremos los distintos elementos que compongan el interfaz. Para mostrar la ventana de controles disponibles (Toolbox), si no estuviera visible, hay que lanzarla desde el menú "Window" en la opción "Toolbox". Una vez visible ya podemos añadir controles a la aplicación seleccionando el control en la toolbox, pinchando en este caso una caja de texto (figura 3). Una vez seleccionado al pasar el puntero del ratón por encima del form este se convierte en una cruz que indica la posición donde situar el control que acaba de seleccionar. Pinche sobre el form y sin soltar el botón, arrastre el ratón hasta que el rectángulo que aparece tenga el tamaño deseado. Al soltar verá como el control se di-



buja al mismo tamaño que el rectángulo anterior. Ahora repita esto mismo pero con otro control, el "command button".

Una vez tenga los dos controles en pantalla observe que pinchando sobre cualquiera de ellos aparecen seleccionados (marcados con rectángulos negros en los lados y esquinas), pudiéndose cambiar tanto el tamaño como la posición del control. Si mientras hay uno seleccionado pincha en otro manteniendo pulsada la tecla "shift" o "ctrl", el segundo también se seleccionará permitiendo así mover los dos a la vez. Este último método (llamémoslo multiselección), no

También se puede cambiar el tamaño del form de la misma manera que se hace en una ventana de Windows

permite modificar el tamaño de varios controles a la vez. También se puede cambiar el tamaño del form de la misma manera que se hace en una ventana de Windows. Una vez situados los controles según el diseño de interface requerido, se puede pasar al siguiente punto del diseño de una aplicación.

Un control que se haya puesto por equivocación, se puede quitar simplemente seleccionándolo y pulsando la techa "Supr" o desde el menú de edición "Edit" en la opción "Delete".

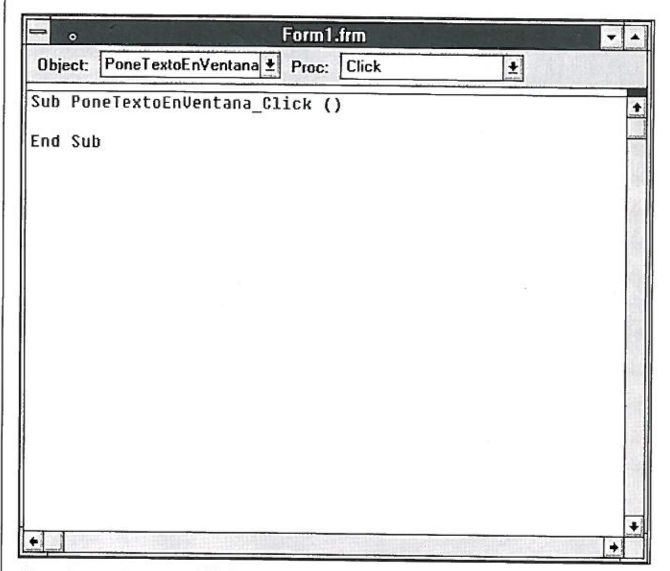
ASIGNACIÓN DE PROPIEDADES

Para la asignación de propiedades, se utilizará la ventana con los controles ya situados y la ventana de propiedades de VISUAL BASIC 3.0 (figura 5). Si esta última está oculta, se puede mostrar de tres maneras diferentes: Lo primero que hay que hacer para cualquiera de las tres, es seleccionar un control pinchando sobre él con el ratón, ya que la ventana de propiedades muestra y permite cambiar las propiedades del control seleccionado en ese momento. Una vez seleccionado el control, se puede pulsar "F4" o bien pinchar sobre el sexto botón de la botonera.mostrándose las propiedades del nuevo control (figura 2).

Pulse sobre el fondo del form para que aparezcan las propiedades de éste. En la ventana de propiedades aparecen dos columnas (figura 5), la de la izquierda es el nombre y la de la derecha, el valor actual. No todas las propiedades se pueden modificar, ya que las hay sólo de lectura o de lectura/escritura. La utilización de las propiedades depende de cual de los dos modos posibles se esté empleando. El primero, "Design mode", es el tiempo en el que se está diseñando la aplicación (en cualquiera de las tres fases antes mencionadas). El segundo es el modo ejecución, cuando el programa está siendo interpretado. En este último modo, las propiedades hay que cambiarlas mediante código. Hay algunas que necesariamente tienen que ser leídas o modificadas en tiempo de ejecución ya que en tiempo de diseño no contienen datos válidos.

Después de esta explicación ya está preparado para cambiar algunas de las propiedades de los controles previamente diseñados.

Utilice la barra de desplazamiento "Slider" de la ventana de propiedades para localizar la propiedad "Caption". En este caso "Caption" es el contenido de la barra de título del *form*. Pinche en la fila correspondiente a esta propiedad para seleccionarla, introduzca el nuevo titulo en la zona de texto que hay en la arte superior de la ventana de propiedades y teclee "Primera aplicación". Como puede comprobar ha cambiado el texto de la barra de título de la ventana. Seleccione con el ratón la caja de texto que Ud. ha creado en la aplicación y observe que la ventana de propiedades, que contiene las propiedades de este



En esta ventana es el único sitio donde se escribe código en Visual Basic.

control y no del *form*. Ahora se cambiará la propiedad text de esta "TextBox" y se dejará en blanco para que al inicio aparezca "limpia". De este control también modificaremos la propiedad "Name" esta propiedad hace referencia al nombre con el que nos referiremos al control en nuestro código. Llegados a este punto, cambiará las propiedades del botón seleccionado, recuerde que ha de seleccionar el botón con el ratón para que aparezcan sus propiedades. Modifique la propiedad *Name* escribiendo "PoneTextoEnVentana" en lugar de "Command1" como aparece en la ventana. Los botones han de ser nombrados de forma que hagan referencia a la acción que estos desempeñan como Borrar, salir, etc.

Si se fija en el texto que aparece en el botón, verá que pone "Command1", para cambiar este texto hay que modificar la propiedad "Caption" del botón por ejemplo escriba "Poner texto".

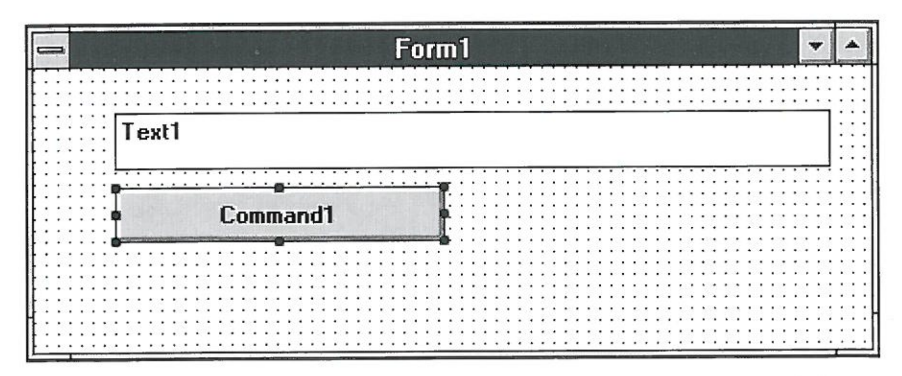


FIG. 7. Esta es la definición de la ventana del ejemplo que se encuentra en el disco.

Un consejo muy importante, no cambie nunca el nombre de un control al que ya hayan asignado código, el código habría que recuperarlo de un lugar que no le corresponde como es el espacio reservado para las declaraciones de funciones y procedimientos.

Una vez el interface ha quedado preparado para recibir código (figura. 8) pasamos a la siguiente fase.

AÑADIENDO CODIGO

Para la adición de código utilizaremos la ventana con los controles en los que queramos situar el código. También es posible utilizar la ventana de proyecto.

Conviene tener muy claros los conceptos de evento y mensaje: Un mensaje es en realidad una llamada a un procedimiento. Si un procedimiento está implementado dentro de un objeto, la llamada se convierte en un mensaje a este objeto. Un evento es la reacción que causa un mensaje en un objeto, estas "reacciones" son las que se deben programar. También es muy importante saber

Para cambiar una propiedad de un control en tiempo de ejecución hay que hacer referencia a ese control

de dónde proceden los mensajes que se convierten en eventos. Estos pueden proceder del usuario por ejemplo al hacer un click de ratón sobre un botón del sistema cuando se carga en memoria una ventana, o también del código que se ha escrito. Después de esta introducción a los eventos se puede pasar a localizar los "sitios" donde se introducirá el código.

Si se hace un doble click encima de cualquier control aparecerá la ventana de código antes explicada (figura 6), esta ventana no es más que un interfaz que ayuda a programar, teniendo separados los controles y eventos de cada control.

Haga ahora doble click sobre el botón que contiene "Poner texto", aparecerá la ventana de código con el siguiente texto: "*Sub Pone- TextoEnVentana_Click ()*" y abajo "*end sub*". *_Click* indica que este procedimiento será llamado cuando el usuario haga un click con el ratón sobre el botón. Sólo queda escribir entre la cabecera del procedimiento y el "end sub", el código que indique lo que se quiere hacer cuando se pulse el botón. En este caso pondremos un texto en el control de tipo "text" que hemos situado en el interfaz. En BASIC clásico ahora se impondría un "print 'texto'", pero en este caso como el control de tipo texto es un objeto, no podemos acceder directamente a su contenido, sino que tenemos que hacerlo mediante sus propiedades.

Para cambiar una propiedad de un control en tiempo de ejecución hay que hacer referencia a ese control. En este caso sería "VentanaDeTexto.text" la primera parte identifica al control y la segunda parte después del punto la propiedad. Para modificarla la tratamos como una variable asignándola en este caso un texto : VentanaDeTexto.text = " Se ha pulsado el botón" (figura 9), las comillas en el texto indican que es un literal, es decir que es exactamente eso lo que queremos que ponga. Cuando se pulse el botón en tiempo de ejecución esta asignación tendrá lugar y en el control de tipo texto aparecerá el literal.

Esta sencilla aplicación no va a contener más código pero si se fijan es totalmente funcional.

Para ejecutar una aplicación en modo intérprete (el antiguo RUN) se puede hacer de dos formas: pulsando "F5" o desde el menú "Run", seleccionando "Start". VISUAL BASIC 3.0 tiene una opción que permite grabar el proyecto antes de cada ejecución (siempre que se haya modificado algo), esta opción está en el menu "Options" en "Environment" y aparece como "Save proyect before run". Si todo ha ido bien la ventana de texto aparecerá vacía y al pulsar el botón el texto ocupará la pantalla.

Para abandonar la aplicación lo que debe hacer es cerrar la ventana como cualquier aplicación Windows. También se puede cerrar desde el menú "Run" de VISUAL BASIC 3.0 seleccionando "End".

GRABACION DE PROYECTOS

Para grabar o salvar una aplicación en VISUAL BASIC 3.0 se deberá salvar independientemente cada uno de los módulos que lo componen. Antes conviene saber que es cada uno de estos:

Los módulos ".VBX" incluidos en el proyecto no se grabarán ya que el programador de VISUAL BASIC 3.0 no puede modificarlos

Los Módulos ".FRM", "forms", son lo que conocemos como ventanas de interface y se guardan con su código en un fichero que podrá ser en formato binario o texto. Se recomienda el formato texto ya que el formato binario sólo puede ser abierto por VISUAL BASIC 3.0.



Los Módulos ".BAS", "Modules", son "almacenes de código" donde declararemos variables, procedimientos y funciones globales.

El fichero de Proyecto ,".MAK", contiene referencias a los módulos que componen el proyecto además de las preferencias del usuario sobre el entorno de desarrollo.

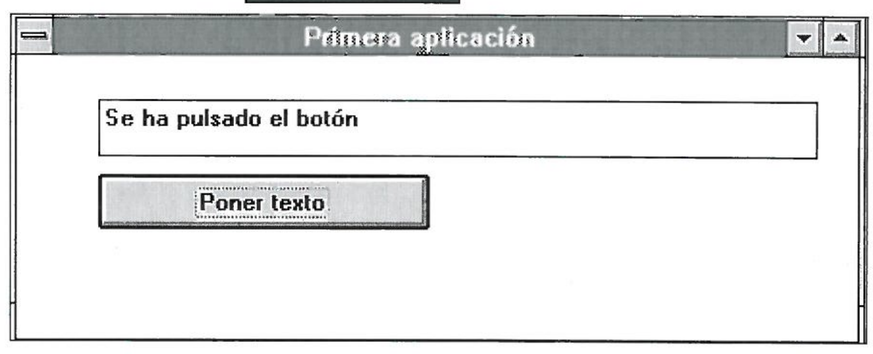


FIG. 8. Ejemplo de la ejecución del programa desarrollado para este artículo.

También pueden aparecer unos ficheros ".FRX" que son gráficos en código binario. Al grabar un ".FRM" que contiene por ejemplo un icono en una de sus propiedades, este no puede ser salvado como texto, por lo que se crea un fichero binario con el icono y desde el fichero ".FRM" en modo texto´se hace referencia al ".FRX". Estos ficheros no aparecerán si se graban los ".FRM" en modo binario ya que la imagen iría incluida sin necesidad de referencias a ficheros externos.

Los programas compilados para ejecutarse han de estar acompañados del fichero VBRUN 300. DLL

CREACIÓN DE EJECUTABLES

Una vez comprobado que el proyecto funciona correctamente es hora de generar un ejecutable ya que sería imposible distribuirlo a no ser que el destinatario del proyecto tuviera VISUAL BASIC 3.0.

Para compilar el proyecto lo único que hay que hacer es decirle a VISUAL BASIC 3.0 que lo haga, desde el menú "File" en la opción "Make Exe File" .



FIG 9. La creación de aplicaciones en Visual Basic necesita muy poco código.

Después de esto puede incluir el ejecutable en cualquier carpeta de Windows, ya que es una aplicación como cualquier otra.

Los programas compilados para ejecutarse han de estar acompañados del fichero"VBRUN300.DLL". Si utiliza ".DLL´s" no estandar, también tendrá que incluirlas con el ejecutable. Toda esta inclusión de archivos y librerías se hace de forma semiautomática una aplicación que incluye VISUAL BASIC 3.0 para generar disquetes de instalación. "Setup Wizard" es la aplicación encargada de esto.

De todas formas, conviene que no se quede en el ejemplo que se ha explicado y profundice por su cuenta en otras propiedades y eventos de los controles ya que aprenderá mucho más.

Aunque al principio al cambiar una propiedad le parezca que no ocurre nada, no es que lo haya hecho mal, se debe a que hay propiedades que producen cambios demasiado sutiles como para apreciarlos a simple vista.

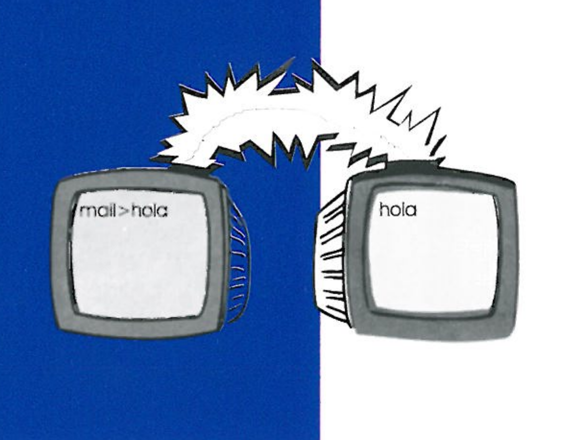
La edición se simplifica bastante con las opciones cortar, copiar y pegar que funciona tanto para código marcando con el ratón o teclas, como para controles en tiempo de diseño.

Como conclusión, se le propone que intente hacer un simple programa que coja el texto de una TextBox y lo deje en otra al pulsar un botón. La única diferencia que existe con el ejemplo que se ha descrito, es el origen del texto a mostrar en la *text box*, que antes era fijo y ahora es introducido por el usuario. Así observará que la propiedad *text* de una *text box* no sólo sirve para poner texto en ella, sino también para recogerlo (es de lectura y escritura). Una vez realizado esto, puede complicarlo tanto como quiera, recogiendo el texto de una text box y mostrándolo en dos o viceversa. Esto ayudará a entender el trabajo que realizan las propiedades de los controles, pero no se quede aquí, experimente con los eventos, escriba código en los eventos como "nombrecontrol.text= "esto es el evento click"", cambie el "click" por el nombre del evento donde lo escriba y "nombrecontrol" por el nombre que Ud. le quiera dar a un TextBox cualquiera, y luego pruebe a pinchar con el ratón, pasar por encima de él, pulsar teclas, etc.

El aprendizaje de VISUAL BASIC 3.0 como de cualquier lenguaje de programación es duro, y más aún, cuando se deja a un lado el diseño clásico Top-Down, pero no lo abandonen ya que una vez pasada la primera fase en la que todo es complicado se pasa a una segunda en la que las horas pasan volando mientras se programa. ■

NETBIOS PRIMERA PARTE

Carlos Arias



Continuando con la serie de artículos dedicados a la programación de entornos de red, vamos a explicar los métodos de implementación de la interfaz NETBIOS.

El interfaz de comunicación NETBIOS (sistema básico de entrada/salida de redes, Network Basic Input Output System), fue desarrollado en 1984 para la tarjeta adaptadora PC Network de IBM y su sistema de red PC Network, más conocido como PC LAN, desarrollado por MICROSOFT. Es por esta razón que los productos de IBM, por ejemplo, PC LAN Server, OS/2, y los de MICROSOFT como LAN Manager y MICROSOFT Windows, utilizan el interfaz común de comunicación NETBIOS.

La fuerza que estos dos "gigantes" de la informática ejercieron sobre el mercado de las redes locales, obligó a otros fabricantes de software a implementar dicho interfaz sobre sus aplicaciones. Este ha sido el caso por ejemplo de NOVELL, que pese ha dominar el 60% del mercado de redes, proporciona un emulador de NETBIOS que se ejecuta sobre los protocolos IPX/SPX. De esta forma el NETBIOS se convierte en el interfaz estándar del nivel de sesión.

COMPONENTES DE NETBIOS

NETBIOS pertenece al nivel de sesión, presentando una conexión orientada a servicio al nivel superior o de presentación. Esto quiere decir que utilizando los servicios NETBIOS el programador no se tiene que preocupar de la gestión a bajo nivel de la red, dejando el control de la corrección de errores, envío de acuses de recibo, etc, a los servicios de las capas más bajas. El NETBIOS utiliza los servicios del nivel de enlace de datos.

TRAMAS

NETBIOS está compuesto por un conjunto de comandos que definen el tipo de estructura de datos transmitidos a la red. Estas estructuras de datos reciben el nombre de tramas de información (Information Frames). La longitud máxima de éstas, depende de la topología de la red, siendo de 1500 bytes para la Ethernet.

NETBIOS puede manejar bloques de datos entre 0 y 64 Kbytes para transferir, fragmentando la información en tantas tramas como sea necesario para su envío y recepción.

NETBIOS maneja dos tipos de trama (véase figura 2) que se describen a continuación:

Tramas de Información: Information Frames o tramas I. Se utilizan para el envío secuencial de tramas entre

dos estaciones de trabajo. A estas tramas NETBIOS les asigna un número ordinal para saber la secuencia de envío. Un ejemplo es la transmisión de ficheros, en donde es importante el orden en que se envían y reciben las tramas de cara a recomponer la información que se va recibiendo en la estación de destino, así como garantizar la correcta recepción de todas ellas. De la posible retransmisión de alguna de las tramas se encarga el nivel LLC (Control de Enlace Lógico, Logical Link Control). Este tipo de trama no se utiliza para enviar mensajes a un grupo o a todos los usuarios de la red, ya que se necesitaría establecer una conexión con cada una de ellas y retransmitir el mensaje tantas veces como conexiones se tengan que hacer.

Para poder utilizar este tipo de tramas es necesario establecer previamente una conexión entre las dos estaciones de trabajo que se van a comunicar.

Tramas de Información No Numerada: Unnumbered Information Frames o tramas UI. Se utilizan, entre otros fines, para enviar información a distintas estaciones de trabajo a la vez mediante la utilización de datagramas, no estando numeradas secuencialmente. La correcta recepción de los datos no está garantizada al no existir acuse de recibo de la estación receptora.

No hace falta establecer conexión previa entre las estaciones de trabajo. Aquellas que reciban la trama, la procesarán internamente y continuarán con su trabajo.

CABECERA DE LA TRAMA

Para poder enviar y recibir tramas, NETBIOS añade una cabecera de tamaño variable según sean tramas I o UI, en la que se indica entre otros datos: la longitud de la trama incluyendo la cabecera, un delimitador formado por dos bytes conteniendo el valor EFFFH, un byte con el comando NETBIOS, número de sesión de destino formado por un byte en tramas I y de dieciséis bytes en tramas UI, número de sesión de origen formado por un byte en tramas I y de dieciséis bytes en tramas UI. Además contiene una serie de datos variables según el tipo de comando que se indique. La descripción detallada de la cabecera se puede observar en la figura 1.

TABLA DE NOMBRES

Es una estructura de datos que contiene información referente a las direcciones de cada estación en particular.

En el primer artículo dedicado a redes locales (ver "Solo Programa dores" del mes de Diciembre), se indicaba que cada tarjeta de red local tiene una dirección física asignada por el fabricante, que identifica de manera inequívoca a la estación de trabajo que la tiene conectada. NETBIOS asigna nombres lógicos de 16 caracteres de longitud a la dirección de cada tarjeta. De esta manera, se consigue una cierta independencia del hardware trabajando con dichos nombres cada vez que se establece una conexión o se envía y recibe información, no importando la dirección física que tenga la tarjeta de la estación emisora o receptora. Así, si en una estación de trabajo cambia la dirección física de la tarjeta, porque ésta se tenga que reemplazar por otra, su dirección o nombre lógico sigue siendo el mismo.

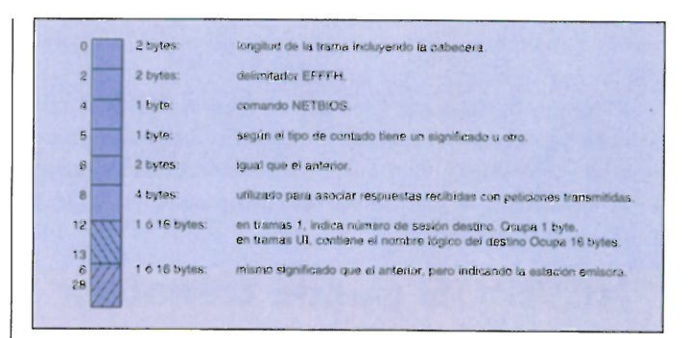


FIGURA 1. Cabecera de una trama NETBIOS.

NETBIOS permite asignar un total de 254 nombres lógicos a cada tarjeta adaptadora. De esta manera, se pueden tener abiertas 254 sesiones a la vez. También permite tener más de una tarjeta a cada estación de trabajo. En este último caso, un ordenador con dos tarjetas puede formar parte de dos redes locales independientes, teniendo asignado el mismo nombre lógico en las dos tarjetas, no comunicándose por este hecho los procesos que ocurren en cada red.

La comunicación síncrona no está "bien vista"

Con la utilización de nombres lógicos en vez de direcciones físicas, un programa puede ejecutarse desde cualquier estación de trabajo, ya que el nombre lógico que se utiliza no tiene nada que ver con la dirección de la tarjeta adaptadora. Si un programa utilizase la dirección física 438723876543h para enviar datos, sólo se podrá comunicar con el ordenador cuya tarjeta adaptadora se corresponda con esa dirección.

Los nombres de NETBIOS pueden ser individuales o de grupo. Los individuales deben ser únicos en la red, no pudiendo existir dos estaciones que compartan el mismo nombre. Los nombres de grupo se utilizan para enviar mensajes a varias estaciones de trabajo a la vez mediante la comunicación no orientada a conexión.

Cuando una aplicación desea añadir un nombre a la tabla, NETBIOS mira que no exista previamente, devolviendo en caso de éxito un número asociado al nombre comprendido entre 1 y 254, que será el utilizado en sucesivas comunicaciones con tramas I.

El Network Control Block o Bloque de Control de Red, es la estructura de datos mediante la cual se envían los comandos al NETBIOS. Su longitud es de 64 bytes y es utilizado conjuntamente con el sistema de llamada al controlador.

La estructura que tiene es la siguiente:

- Comando NETBIOS: comando NETBIOS a procesar. Este comando se puede procesar de manera sín-

crona o asíncrona. Estas dos formas de proceso se explican más adelante en el artículo.

- Código de retorno: "semáforo" que indica si el comando ha sido procesado con éxito. En caso afirmativo contiene el valor 0. Con comandos asíncronos un valor FFH indica que el comando ha sido puesto en espera para que se procese.

NETBIOS puede transferir bloques desde 0 a 64 Kbytes de información

- Número de sesión local: indica el número de sesión local. Se obtiene del controlador NETBIOS al establecer una sesión. En las comunicaciones posteriores que se realicen, se puede indicar este número en vez de utilizar el nombre lógico de la estación remota.

- Número del nombre: es un número asignado por el controlador de NETBIOS cuando se añade un nombre a la tabla de nombres.

- Dirección del buffer: zona de memoria en la que se encuentran los datos a enviar, o donde se han de colocar los datos recibidos.

- Longitud del buffer: si el buffer es de envío de datos, lo fija el programa. Si el buffer es de recepción, la longitud la pone NETBIOS.

- Nombre lógico de la estación remota: tiene una longitud de 16 bytes. Es rellenada por NETBIOS cuando la estación está esperando recibir una petición de conexión por parte de una estación remota. Si es la propia estación la que establece la conexión o si envía un datagrama, es la aplicación la encargada de rellenar este campo.

- Nombre lógico de la estación local: tiene al igual que la anterior, 16 bytes de longitud, y es el nombre lógico de la estación. Es la aplicación la que rellena este campo cuando establece una conexión o envía un datagrama.

- Tiempo en recibir: es el tiempo que hay que esperar, en periodos de medio segundo, la recepción de un paquete de uno o varios procesos remotos. Si transcurrido este tiempo no se ha recibido ningún paquete, el código de retorno del NCB es puesto a un valor de error. Si se especifica un valor igual a cero, NETBIOS bloquea la ejecución hasta que un paquete es recibido.

- Tiempo en enviar: Indica el tiempo que hay que esperar, en periodos de medio segundo, antes de que un comando como Send de por finalizado los intentos de enviar información. Un valor de 0 hace que NETBIOS se quede bloqueado hasta que se consiga enviar la información con éxito.

- Dirección de la rutina de servicio de eventos: utilizada por los comandos en modo asíncrono. Apunta a una rutina de tratamiento de la información recibida cuando el comando NETBIOS a finalizado. Indica la dirección del segmento y el offset de la rutina.

Para establecer una comunicación se crea un Circuito Virtual

- Número de Lan: en una estación con más de un adaptador de red, indica cual de ellos se va a utilizar.

- "semáforo" utilizado en el modo asíncrono para indicar cuando un comando se ha terminado de procesar. Un valor FFH indica que el comando no ha sido procesado por completo todavía.

Para realizar llamadas al controlador de NETBIOS bajo MSDOS, se utiliza la interrupción 5CH, pasando en los registros ES y BX, el segmento y el desplazamiento de la dirección de la estructura NCB.

Nombre de la trama	Tipo
ADD_GROUP_NAME_QUERY	UI
ADD_NAME_QUERY	UI
NAME_IN_CONFLICT	UI
STATUS_QUERY	UI
TERMINATE_TRACE	UI
DATAGRAM	UI
DATAGRAM_BROADCAST	UI
NAME_QUERY	UI
ADD_NAME_RESPONSE	UI
NAME_RECOGNIZED	UI
STATUS_RESPONSE	UI
TERMINATE_TRACE	UI
DATA_ACK	I
DATA_FIRS_MIDDLE	I
DATA_ONLY_LAST	I
SESSION_CONFIRM	I
SESSION_END	I
SESSION_INITIALIZE	I
NO_RECEIVE	I
RECEIVE_OUTSTANDING	I
RECEIVE_CONTINUE	I
SESSION_ALIVE	I

FIGURA 2. Tipos de trama NETBIOS.

SERVICIOS DE NETBIOS

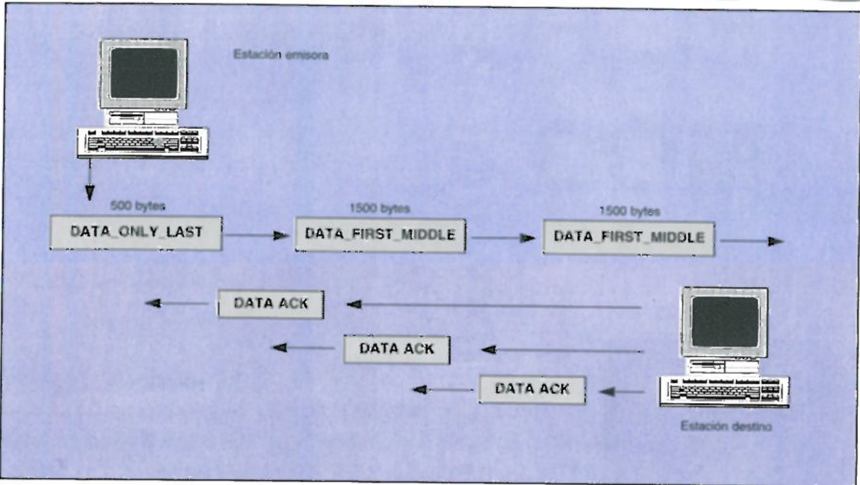
El método de conexión de la aplicación con el controlador NETBIOS varía según el sistema operativo que se utilice. En MSDOS normalmente se utilizan las interrupciones 5CH y 2AH.

Las funciones de servicio que proporciona NETBIOS se dividen en las siguientes categorías:

Mantenimiento: son las encargadas de añadir, borrar un nombre en la tabla de nombres, etc.

Transferencia de Datos Orientada a Conexión: utilizadas para crear un circuito virtual, explicado más adelante, entre dos estaciones de trabajo. Se encargan de establecer, mantener y liberar una conexión.





Tramas que se envían para transferencia de un fichero.

Transferencia de Datos No Orientada a Conexión: se utilizan para la difusión de un mensaje a parte o todos los miembros de la red. Propósito general: proporciona servicios para averiguar el estado del adaptador de red, busca un nombre lógico en la red devolviendo cierta información sobre su localización, cancelar una operación, etc.

En la 2º parte de este artículo, que aparecerá el mes que viene, se detallará la utilización de los comandos que tiene disponibles los distintos servicios de NETBIOS.

TIPOS DE CONEXIONES

Dependiendo de la finalidad de la comunicación y de las tramas que se utilizan para llevarla a cabo, NETBIOS utiliza dos tipos de transferencia: la Transferencia de datos Orientada a Conexión, y la Transferencia de datos No Orientada a Conexión.

TRANSFERENCIA DE DATOS ORIENTADA A CONEXIÓN

La Transferencia de Datos Orientada a Conexión se utiliza cuando se necesita que el envío de información sea fiable o hay que transmitir paquetes de datos que ocupan más de una trama, siendo importante que el orden de recepción coincida con el de envío. Este es el caso de la transferencia de ficheros.

Para llevar a cabo la conexión, NETBIOS crea un mecanismo llamado circuito virtual. Los nombres de las estaciones de trabajo que establecen este circuito virtual no pueden ser de grupo. Si la información a transferir es mayor que la longitud de una trama, NETBIOS la divide en tantas tramas como sea necesario.

Una vez que ha sido establecido el circuito con éxito se pueden utilizar tramas I para la transferencia de datos. Las tramas que se utilizan son las siguientes:

Data_First_Middle: se utilizan para enviar un paquete de datos que no cabe en una sola trama. Todas las tramas que se envían son de este tipo, excepto la última, que es de tipo Data_Only_Last.

Data_Only_Last: se utiliza para indicar que es la última trama de una secuencia de datos o cuando el paquete de datos a enviar cabe en una sola trama.

Data_Ack: es enviada por la estación receptora para indicar que ha recibido la información correctamente.

TRANSFERENCIA DE DATOS NO ORIENTADA A CONEXIÓN

La transferencia de datos no orientada a conexión se denomina Datagrama. Se utiliza para enviar información a un grupo de estaciones al mismo tiempo. Con este tipo de transferencia el destinatario no envía ningún acuse de recibo. Por tanto, no se puede asegurar que la información ha sido recibida correctamente.

Las tramas que se utilizan para la transferencia de información son del tipo UI. Estas son:

Datagram: se utilizan para enviar información a una estación de trabajo o a un nombre de grupo. En ambos casos se especifica el nombre lógico al que se va a transmitir la información.

Datagram_Broadcast: es una trama especial, ya que no se especifica ninguna dirección lógica, transmitiendo la información a todos los puestos de la red. En este caso, también recibe la información el ordenador que la envió.

GESTION DE LA RECEPCION DE PAQUETES

NETBIOS tiene la posibilidad de gestionar el envío de información, bien de forma asíncrona o bien de forma síncrona.

CONEXION ASINCRONA

Es el tipo de conexión "bien visto" y el más comúnmente empleado. La estación de trabajo ejecuta un comando NETBIOS y continúa con la ejecución del programa sin esperar a que NETBIOS termine de procesar el comando. A intervalos regulares se examinan el estado de los "semáforos" ncb_cmd_cplt y ncb_retcode, incluidos en el NCB, que indican si el tiempo transcurrido entra dentro de los márgenes de espera y si el comando ha sido ejecutado con éxito. De esta manera la aplicación puede seguir realizando otras tareas sin pérdida de tiempo.

CONEXION SINCRONA

ejecuta un comando de NETBIOS, y éste no devuelve el control hasta que el comando ha sido procesado. Este tipo de conexión no está "bien visto", ya que el programa pierde tiempo mientras espera la confirmación de la estación destino y puede llegar a dejar "colgado" a la estación de trabajo. ■

EL FORMATO MOD

Luis Crespo

Los ficheros MOD están de moda: rara es la publicación que incluya un CD-ROM que no contenga montones de estos ficheros para poder rellenar los 500-600 MB del CD. Originalmente, los módulos musicales se utilizaban en el Amiga, pero la evolución de los PCs en velocidad de proceso y en hardware de sonido ha hecho posible que también puedan ser interpretados por éstos.

Los MODs siempre han sido soportados por amateurs, mientras que las empresas y profesionales del software los han estado ignorando durante mucho tiempo. Prácticamente todos los programas de composición o reproducción de módulos musicales son shareware o freeware.

Originalmente, los módulos musicales se utilizaban en el Amiga

Los fuentes correspondientes a esta entrega consisten en un programa que carga un fichero MOD en memoria, muestra sus datos por pantalla y permite hacer sonar los instrumentos a distintas frecuencias de muestreo, utilizando las rutinas de sonido presentadas en las entregas anteriores. El programa principal se encuentra en el fichero MODTEST.C o MODTEST.PAS.

¿QUE ES UN MOD?

Un MOD es un fichero que contiene una composición musical, dividido en tres partes muy diferenciadas:

1. Cabecera: contiene datos como el título de la canción, nombre y tamaño de los instrumentos, etc.

2. Partitura: es la sucesión de notas musicales, que convenientemente interpretadas resultarán en la melodía.

3. Instrumentos o samples: no son más que sonidos digitalizados, es decir, secuencias de muestras como las que se han estado tratando en las dos entregas anteriores de esta serie de artículos. Cuando suena una nota de la partitura, ésta indica entre otras cosas qué instrumento, (es decir, qué digitalización) debe sonar.

El hecho de que la música generada esté basada en digitalizaciones da una gran versatilidad a este formato, y es la clave de su éxito. A diferencia de otros formatos

Diseñados originalmente para adaptarse al hardware del ordenador Amiga, los módulos musicales (ficheros MOD) pueden interpretarse desde gran variedad de hardware y suelen ser muy espectaculares.

musicales afortunadamente ya venidos a menos (CMF, ROL, etc), la variedad de los instrumentos es infinita y el realismo es prácticamente total. Un golpe de tambor, por ejemplo, suena tal como tiene que sonar y no como una triste aproximación sintética. Además, al incluir cada módulo sus propios instrumentos, cada canción tiene su propia "personalidad" sonora, lo cual da mucho juego a la creatividad del autor.

Las notas canales se juntan en grupos de 64 para formar un patrón

El ordenador Amiga tiene cuatro DACs que permiten reproducir cuatro digitalizaciones distintas simultáneamente, cada una a distinta frecuencia de muestreo. Por eso, los MODs contemplan un máximo de cuatro notas simultáneas, y dividen la partitura en cuatro canales, cada uno de los cuales se corresponde con un DAC del Amiga. Como por cada canal sólo puede estar sonando una nota de un instrumento en cada momento, cada nueva nota anula la que estaba sonando anteriormente en ese canal.

Como la mayor parte de las tarjetas de sonido de PC tienen un sólo canal (o dos si son estéreo), un programa que interprete módulos deberá mezclar los canales por software para emular varios canales de sonido utilizando realmente sólo uno. La mezcla software, además, deberá tener en cuenta en la emulación que cada canal puede funcionar a distintas frecuencias de muestreo.

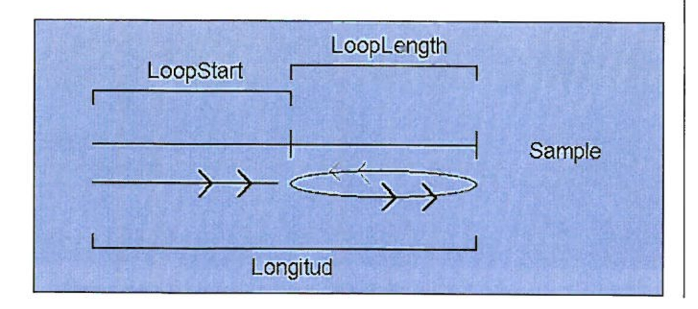
LA CABECERA

La tabla 1 contiene un esquema de la cabecera con cada campo y sus correspondientes longitudes. A continuación se describe con detalle el significado de cada campo:

1. Titulo: Título de la canción, en formato ASCIIZ (es decir, ASCII terminado en 0), a no ser que el texto ocupe hasta el último carácter que tiene reservado, en cuyo caso no hay ningún 0.

2. Instrumentos: Datos correspondientes a cada instrumento, repetidos 15 o 31 veces. Cada instrumento tiene los siguientes campos:

- Nombre: Nombre o descripción del instrumento, en el mismo formato que el título de la canción. Muchos autores aprovechan estos espacios para poner comentarios como su nombre, fecha de composición, o saludos a amigos, miembros de grupos o incluso dedicatorias amorosas.



- Longitud: Longitud de la digitalización, en words. A esto último es necesario hacer un par de aclaraciones. Como los ficheros MOD fueron diseñados originalmente para ser utilizados por los ordenadores Amiga, hay una diferencia de criterio en la forma de almacenar los números: mientras los ordenadores como el Amiga (y en general todos los que utilizan un microprocesador de la familia del Motorola 68000) almacenan los números en memoria empezando por los bytes más significativos, en los ordenadores con microprocesador de la familia del Intel 80x86 se almacenan al revés, es decir, empezando por los bytes menos significativos.

La cuestión es que cuando se carga con el PC un número que ocupe más de un byte, ese número estará "dado la vuelta". De esta forma, debe intercambiarse el orden de los bytes de todos los datos que hacen referencia a las longitudes del sample correspondiente a cada instrumento. Además, como las longitudes vienen expresadas en words (2 bytes), es necesario multiplicarlas por 2 para conocer el valor en bytes.

Según esto, podría ocurrir que la longitud de un sample sea superior al tamaño de un segmento, en cuyo caso sería necesario darle un trato especial a la hora de cargarlo y reproducirlo en memoria (o bien utilizar modo protegido de 386). Pero en la práctica ningún MOD utiliza samples de más de 64 Kb, de forma que puede ignorarse este problema o incorporar una rutina que detecte este caso especial y de un mensaje de error.

- Finetune: Traducido del inglés, significa "afinamiento preciso". Este campo aporta información sobre la forma de hacer sonar las notas del instrumento, afinando sutilmente el tono de cada nota. Sólo los 4 bits más bajos del byte contienen datos, en complemento a dos:

0-7 : Afinamiento positivo. La nota debe subir de tono x/8 semitonos, siendo x el valor del Finetune.

8-15: Afinamiento negativo. La nota debe bajar de tono (16-x)/8 semitonos, siendo x el valor del Finetune.

Supuestamente, el finetune es modificado por los músicos que conjuntan varios instrumentos y quieren que todo suene a la perfección. En la práctica, este valor raras veces es distinto de 0, y por lo tanto pocos MODs van a sonar distinto si no se tiene en cuenta esta información.

- Volumen: Es el volumen por defecto del instrumento, es decir, el volumen con el que sonarán todas las notas de ese instrumento a no ser que se especifique lo contrario. Sus valores permitidos van de 0 a 64.

- LoopStart y LoopLength: Como grabar un sonido de larga duración gasta mucha memoria y espacio en disco, el loop (bucle) es un sistema para evitar este

Nombre	Tamaño	Descripción
Título	20	Título de la canción
Repetido 15 o 31 veces:		
Nombre	22	Nombre del instrumento
Longitud	2	Tamaño de la digitalización
Finetune	1	Afinamiento preciso
Volumen	1	Volumen por defecto
LoopStart	2	Inicio del "loop"
LoopLength	2	Longitudo del "loop"
NumSeq	1	Entradas válidas en la sec.
JumpPos	1	No utilizado
Secuencia	128	Tabla de nº de patrones
Identificador	4	Marca de tipo de fichero
Patterns	?	Secuencia de notas
Samples	?	Digita. de los instrumentos

TABLA 1: Cabecera de un fichero MOD.

problema en algunos casos. En bastantes instrumentos, a partir de cierto instante el sonido se puede prolongar casi indefinidamente sin apenas alterarse, como es el caso de un violín o de un sonido suave de sintetizador. De forma que, con grabar un instante del sonido y repetirlo cuantas veces haga falta, puede conseguirse el efecto de hacer que un instrumento sea de larga duración. Cuando se hace sonar un instrumento con "loop" (ver figura 1), se recorre el sample desde el principio del mismo hasta el final del loop, y cuando se llega a ese punto se vuelve a recorrer, esta vez desde el principio del loop (LoopStart) hasta el final del loop (LoopEnd), y se repite ese bucle hasta que una nueva nota cancela la que estaba sonando.

De nuevo hay que hacer una serie de manipulaciones con estos datos para que puedan ser utilizados adecuadamente. En primer lugar hay que intercambiar la parte alta con la parte baja de cada word y multiplicar por 2. Y en segundo lugar, en el fichero viene primero la posición de inicio de loop y luego la longitud del loop. Pero es más conveniente conocer la posición del final del loop, para evitar tener que sumar cada vez, de forma que se calcula la posición del final del loop (LoopEnd) a partir de LoopStart y LoopLength. Además, muchos MODs tienen errores o diferencias de criterio en estos campos, y hay que comprobar que los datos sean correctos, y corregirlos en caso contrario. De todo esto se encarga la rutina ConvertInst() del fichero MODLOAD.C (y MODLOAD.PAS).

Como ya se ha dicho, los campos que van desde Nombre hasta LoopLength se repiten 15 o 31 veces, y a continuación se encuentran los siguientes campos:

- NumSeq: Número de entradas válidas de la secuencia. Hace referencia a la longitud de la canción, y se explica con detalle más adelante.

- JumpPos: Este dato puede ser ignorado. Al principio se utilizaba para indicar la posición a la cual saltar al terminar la canción, pero actualmente se utiliza otro sistema, como se verá más adelante.

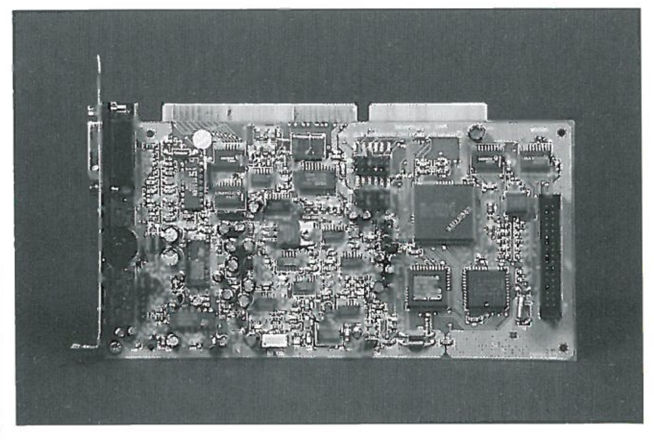
- Secuencia: Es una tabla de 128 posiciones en la que se indica qué patrón tocar en cada momento (explicado más adelante).

La nota musical depende de la frecuencia a la que se reproduce la digitalización

- Identificador: Estos cuatro caracteres son el indicativo del formato MOD. Los primeros módulos tenían un máximo de 15 instrumentos, que posteriormente se amplió a 31. Junto con esta ampliación se incluyó la marca "M.K." en esta posición, haciendo referencia a Mahoney & Kaktus, que fueron quienes introdujeron esta mejora. Más tarde otros programas también utilizaron la extensión a 31 instrumentos y pusieron su propia marca, y de este modo se puede encontrar "FLT4" como la marca que deja el programa de Amiga Startrekker, o "FLT8" si se trata de un módulo de 8 canales. Pero las extensiones al formato original no terminan aquí: ya en el "mundillo" de PC, el programa de composición Fasttracker del grupo sueco Triton permite utilizar 6 u 8 canales, grabando la marca "6CHN" u

Con el identificador se averigua el número de instrumentos y de canales

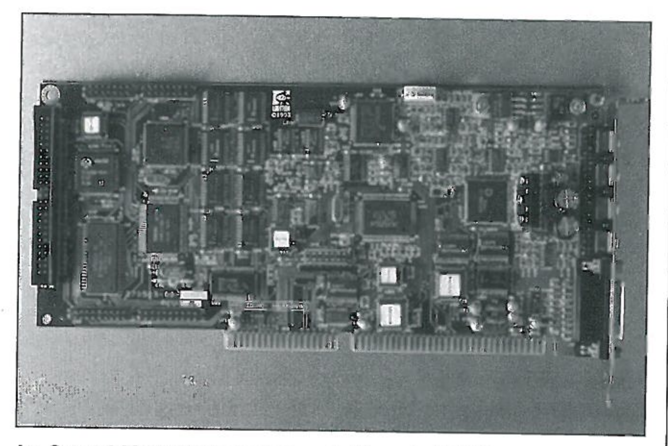
"8CHN" respectivamente. Otros programas de composición para PC también utilizan el formato del Fasttracker, y el propio Fasttracker en su reciente versión II lo amplia hasta 32 canales utilizando la marca "xxCH", donde xx determina el número de canales y puede variar de 10 a 32.



La Sound Blaster Pro es actualmente un estándar en sonido.



Como puede verse, este campo es muy importante, porque se utiliza para averiguar de qué variante se trata. Si no se encuentra ninguna marca reconocible, entonces se puede suponer que se trata de un módulo original (el generado por el programa Noisetracker), y por lo tanto tiene 15 instrumentos en lugar de 31. La rutina GetFormat() se encarga de averiguar el número de canales y de instrumentos en función del identificador.



La Sound Man Wave posee un intérprete MIDI integrado.

Es necesario tener en cuenta que los canales 1 y 4 suenan por la izquierda, mientras que los canales 2 y 3 por la derecha. Si el MOD tiene más de 4 voces, se aplica el mismo criterio: canales 5 y 8 por la izquierda, 6 y 7 por la derecha, etc. La rutina CalcVolume() determina el volumen para cada canal y lo almacena en una estructura de datos utilizada para este propósito.

LA PARTITURA

Justo a continuación de la cabecera se encuentra la información referente a la secuencia de notas. Las notas para cada uno de los 4 o más canales se juntan en grupos de 64 para formar un patrón (en inglés, pattern). Esta agrupación en patrones se utiliza para poder secuenciar más cómodamente la sucesión de notas, de forma que en el programa de composición se puede indicar que se interprete, por ejemplo, primero el patrón 1, luego dos veces el 3, luego el 2, luego de nuevo el 3, etc.

Precisamente eso es lo que indica la tabla de 128 bytes de la cabecera: la secuencia, es decir, el orden en el que deben interpretarse los patrones. De modo que la música comienza por el patrón indicado en la primera posición de la secuencia, continúa por el patrón indicado en la segunda, y así sucesivamente hasta llegar a la última posición válida de la secuencia, que está indicada en la cabecera en el campo NumSeq. De hecho, NumSeq indica el número de posiciones válidas, de forma que si la tabla comienza en la posición 0, la última posición válida será NumSeq-1. Cuando la canción alcance la última nota del patrón de la última posición de la secuencia, la música terminará, o bien volverá a comenzar desde el principio (dependiendo de cada programa).

Byte 1	Byte 2	Byte 3	Byte 4
76543210	76543210	76543210	76543210
YYYYXXXX	xxxxxxxx	yyyyzzzz	tttttttt

XXXXxxxxxxxx	(12 bits)	:	Periodo
YYYYyyyy	(8 bits)	:	Número de instrumento
zzzztttttttt	(12 bits)	:	Comando y parámetro del efecto

FIGURA 2: Estructura de una nota.

Cada nota contiene los siguientes datos:

- Período: Determina la nota musical propiamente dicha. El período es la inversa de la frecuencia: cuanto mayor es el período, más grave es la nota, y cuanto menor es, más aguda.

Este dato viene escalado a conveniencia del hardware de Amiga, de forma que será necesario hacer una serie de manipulaciones antes de poderlo utilizar en el PC. Si el valor de este campo es 0, se trata de una nota "vacía", es decir, la nota anterior de ese canal seguirá sonando.

El Amiga almacena los números empezando por los bytes más significativos

- Número de instrumento: el número de digitalización que debe reproducirse con la nota. Un 0 en este campo significa que debe utilizarse el instrumento que se estaba usando anteriormente (nótese que esto explica por qué el número de instrumentos es una potencia de 2 menos 1).

- Efecto: los efectos son comandos especiales que pueden modificar la nota que está sonando o alterar datos globales como el tempo o el secuenciamiento de la melodía. Un efecto está dividido en dos partes: el comando y su parámetro correspondiente. Por ejemplo, uno de los efectos modifica el volumen por omisión de la nota: el comando es el 12, y el parámetro es el volumen en cuestión.

La mayor parte de las tarjetas de sonido tienen un sólo canal

Cada nota está empaquetada en 4 bytes según se indica en la figura 2. Como puede verse, el número de instrumento ocupa 8 bits, pero sólo deben tenerse en cuenta los 5 bits más bajos ya que el número máximo de instrumentos es 31.

Las notas se almacenan en el fichero consecutivamente una para cada canal, y esto se repite 64 veces

para cada patrón. Por ejemplo, en un MOD de 4 canales, un patrón ocupará:

4 (bytes/nota) * 4 (canales/línea) * 64 (líneas/ patrón) = 1024 bytes/patrón.

El KDMA es un chip especializado en la transferencia de datos

Para determinar el número de patrones que contiene el fichero, hay que buscar en la tabla de la secuencia (desde la primera posición hasta NumSeq), en donde se hace referencia a los números de patrón. El número más alto contenido en la tabla determinará cuál es el mayor patrón utilizado. Como los patrones se numeran a partir del 0, el número obtenido más uno indicará el total de patrones almacenados en el módulo. De ello se encarga la función GetNumPatts().

LOS SAMPLES

Tras el último patrón comienzan los datos correspondientes a las digitalizaciones de cada instrumento, en el mismo orden que vienen dados en la tabla de 15 o 31 entradas de la cabecera. El número de muestras (bytes) a leer por cada instrumento viene dado por el campo de longitud (una vez ha sido convenientemente arreglado). Si un instrumento tiene la longitud a 0, significa que no tiene ninguna digitalización asociada y por lo tanto debe ignorarse.

Las digitalizaciones están almacenadas en el formato de 8 bits y con signo, es decir, cada muestra puede tomar cualquier valor entre -128 y +127. En el PC, la mayor parte de dispositivos de sonido utilizan el formato sin signo, de modo que cada muestra puede tomar cualquier valor entre 0 y 255. Para convertir las muestras de uno a otro formato basta con sumar 128 a cada una. De todas formas, esto sólo será necesario en el caso de que se quiera reproducir una digitalización de forma individual. A la hora de mezclar varias digitalizaciones, es más conveniente manejar las muestras con signo, como se verá en la siguiente entrega.

TEMPORIZACION

Una vez visto cómo se almacenan los módulos en fichero, es necesario explicar cómo deben ser interpretados. Una cuestión muy importante es la relativa a cuándo hacer sonar cada nota.

De nuevo por cuestiones del hardware del Amiga, la base de tiempos que se utiliza es una cincuentava parte de segundo, es decir, 2 centésimas de segundo. A esta base de tempo se le suele llamar tick. Versiones modernas del formato permiten modificar la duración de un tick, pero la mayoría de los módulos utilizan el valor por defecto.

El parámetro Tempo (también llamado Speed en algunos programas) es el que determina el intervalo de tiempo entre cada nota y la siguiente: si el tempo vale 5, por ejemplo, las notas se sucederán cada 5 ticks, o lo que es lo mismo, cada décima de segundo. El tempo por defecto es 6, por lo tanto cada nota durará 0.12 segundos.

La duración de un tick viene determinada por el BPM, que son las iniciales de Beats Per Minute, en castellano "golpes por minuto", que es igual al número de redondas por minuto. Sabiendo que el tempo por defecto es 6, que la duración por defecto de un tick es de 1/50 de segundo, y teniendo en cuenta que entre una nota y la siguiente hay una negra (4 negras forman una redonda), se puede deducir que el BPM por defecto es 125:

50 (ticks/segundo) * 1/6 (nota/tick) * 1/4 (redonda/nota) * 60 (segundos/minuto) = 125 redondas/minuto = 125 BPM.

NOTAS

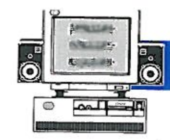
La nota musical viene determinada por la frecuencia a la que se reproduce la digitalización. Si por ejemplo se digitaliza un Do-3 de un piano a 22 KHz, para obtener un Do-4 con la misma digitalización bastará con reproducirla a 44 KHz. En general, cuando se multiplica por dos la frecuencia de muestreo se aumenta en uno la octava de la nota, y cuando se divide entre dos se disminuye en uno la octava. Como hay 12 semitonos en una octava y todos están separados del siguiente por un factor multiplicativo, es fácil deducir que si se multiplica la frecuencia de una nota por la raíz doceava de dos se obtiene la frecuencia del siguiente semitono.

Las digitalizaciones están almacenadas en formato 8 bits y con signo

Sabiendo además que el valor del divisor 1712 corresponde al Do de la octava más baja, se puede construir una tabla que haga corresponder a cada nota musical un determinado período. De esta forma se puede deducir la nota y octava a partir del período, lo cual puede ser útil. Para ello basta con recorrer la tabla hasta dar con el período buscado, y la posición en la que se encuentre determinará la nota. Esto es lo que hace la rutina CalcMusNote(). Hay que tener en cuenta que el período es el inverso de la frecuencia, y por lo tanto, por cada octava que se aumenta el período se divide por dos.

EFECTOS

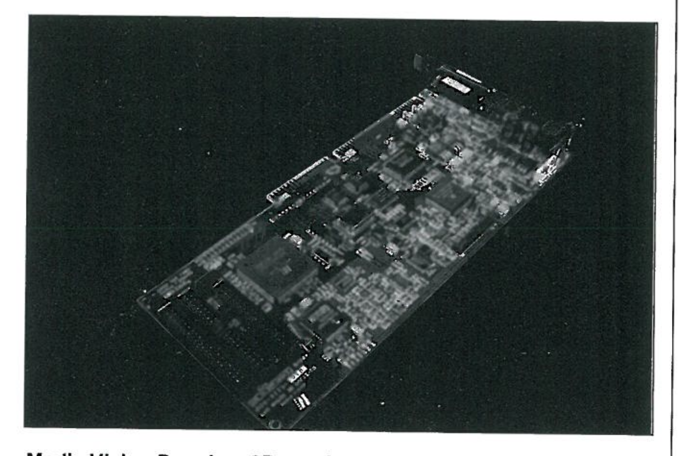
Conforme el formato de los módulos ha ido evolucionando, se han añadido más y más efectos a la lista. Actualmente hay una gran variedad de efectos, pero



muchos de ellos son tan rebuscados o sutiles que es muy difícil, cuando no imposible, encontrar un MOD que los utilice. A continuación se describen los comandos que alteran la forma de recorrer la partitura:

- Comando 15: Cambiar tempo o BPM. Si el parámetro es menor que 32, éste indica el nuevo tempo de la canción, y en caso contrario indica los nuevos BPM.

- Comando 13: Fin de patrón. Los patrones se interpretan normalmente desde la primera hasta la última nota, para pasar a interpretar a continuación el siguiente patrón indicado en la secuencia. Este comando rompe con esta secuenciación, haciendo que la siguiente nota a leer sea del siguiente patrón. De esta forma, por ejemplo, es muy sencillo utilizar un compás de 3/4 a base de colocar este comando en la nota 48 de cada patrón. Normalmente el parámetro de este efecto está a 0, de forma que el siguiente patrón comenzará por la primera nota, pero en general el parámetro indica el número de nota al cual saltar. Como los programas de composición utilizan intensivamente la notación hexadecimal, el formato del parámetro es un tanto extraño: el número debe "escribirse" en hexadecimal e interpretarse a continuación en decimal, es decir:



Media Vision Premium 3D puede generar sonido tridimensional.

```
numnota = (parámetro && 0xF0)*10 + (parámetro && 0x0F);
```

- Comando 11: Salto de posición. Similar al comando 13, el parámetro indica la posición de la secuencia a la cual saltar, posicionándose siempre en la primera nota del nuevo patrón. En este caso el parámetro no necesita ser transformado: su valor es directamente el índice de la tabla de patrones.

Los comandos que modifican parámetros de cada nota como vibrato, portamento, trémolo, etc, se explicarán más adelante en el momento de implementarlos. De todas formas, vale la pena conocer un comando muy básico como es el que modifica el volumen de la nota:

- Comando 12: Cambiar volumen. Modifica el volumen de la nota, de forma que en lugar de utilizarse el volumen por defecto especificado en el instrumento correspondiente, se utiliza el indicado en el parámetro del comando. Los valores posibles van de 0 a 64.

EL PROGRAMA DE EJEMPLO

El programa MODTEST.EXE espera un nombre de fichero MOD como parámetro. Si el fichero se carga correctamente, presenta en pantalla información sobre el módulo y sus instrumentos. Cada instrumento va precedido por un número o una letra, y si está marcado con un asterisco tienen una digitalización asociada y por lo tanto puede hacerse sonar pulsando la tecla correspondiente al número o letra que lo precede. En la parte inferior de la pantalla hay un número que indica la frecuencia de muestreo a la que se reproducirán las digitalizaciones. El valor de la frecuencia se puede variar pulsando las teclas + y -. Para salir del programa hay que pulsar la tecla ESC.

El parámetro *Tempo* determina el intervalo de tiempo entre notas

En los fuentes que acompañan este artículo se utiliza una estructura de datos llamada TSong que se encarga de albergar todos los parámetros y datos de la canción. La rutina que carga un módulo a memoria se encarga de convertir la estructura interna de un MOD a la del TSong, adelantando trabajo a la hora de interpretar la partitura. Por ejemplo, se utiliza una estructura de datos llamada TNote que contiene todos los parámetros referentes a una nota, y la rutina de carga "desempaqueta" las notas del fichero, las interpreta y distribuye convenientemente cada campo en la estructura. Como la estructura TNote tiene un campo dedicado a contener el volumen, la rutina que convierte una nota comprueba si el comando es el 12 y en caso afirmativo modifica el volumen de la nota.

El hecho de utilizar esta estructura tiene otra ventaja: el formato MOD es el formato más extendido de música digital, pero no es ni mucho menos el único. Otro formato muy famoso es el S3M, que es el de los ficheros que genera el programa Scream Tracker 3. La estructura TSong pretende ser lo bastante genérica como para poder cargar distintos tipos de ficheros de música.

EN LA PROXIMA ENTREGA

En la próxima entrega de la serie se explicará cómo mezclar varios canales en uno sólo y cómo modificar independientemente la frecuencia de muestreo y el volumen de cada canal. Se presentará una primera versión de un intérprete (player) de ficheros MOD, que si bien no implementará la mayoría de efectos, ya podrá hacer sonar cualquier módulo. ■

POR LA PUERTA GRANDE

Agustín Guillén

El microprocesador necesitaba de algún sistema o mecanismo para comunicarse con los periféricos que le rodean, y los ingenieros que lo diseñaron dotaron al PC de los Puertos de Entrada y Salida.

Los puertos son la vía de conexión entre los distintos periféricos que rodean al PC y el microprocesador, y mediante ellos es posible controlar, manejar y determinar el estado de los dispositivos de entrada y salida.

El espacio de memoria para los puertos consiste en 64 Kbytes, que pueden ser divididos en 64 Kb de puertos de 8 bits o intercalando puertos de 16 bits o de 32 bits, siempre sin exceder los 64 Kbytes disponibles. Estos 64 Kb son memoria física, pues su acceso está exento de la paginación o segmentación, incluso en procesadores de 32 bits. Los puertos entre la dirección 00F8H y la 00FFH son para uso reservado de Intel y no los utilizan ningún periférico.

Los puertos adyacentes suelen estar emparejados y trabajan conjuntamente, de manera que con el de la dirección más baja se selecciona un índice para indicar qué función o parámetro se desea escribir o leer, y con el segundo se transfiere el valor en cuestión. Esto permite comprimir en una sola pareja de puertos muchas y

El espacio de memoria para los puertos consiste en 64 Kbytes

variadas funciones. Además, no se ralentiza el acceso, pues sólo hay que acceder como si fuera un puerto de 16 bits y el hardware realiza el resto. Unos ejemplos claros de esto son: el acceso a la memoria RAM del CMOS (puerto 0070 registro índice, puerto 0071 datos), el registro CRT de la tarjeta de vídeo (03B4 índice, 03B5 datos), o el registro TS también de la tarjeta de vídeo (03C4 índice, 03C5 datos).

Las instrucciones que el microprocesador tiene implementadas para acceder a los puertos son las siguientes:

- IN: Transfiere un byte, word o doble word de datos, desde el puerto especificado por el segundo operando en el registro indicado por el primer operando (AL, AX, EAX). Para acceder a cualquiera de los 65535 puertos, se debe cargar el registro DX con el número de puerto y pasarlo como el segundo parámetro. Sólo se podrá acceder a un puerto directamente (sin cargar su número en DX) si su número es menor de 256 (8 bits).

Existen macros o pequeñas funciones para ofrecer la capacidad de acceso a los puertos

Ejemplos:

IN	AL, 40h;	Carga en AL un byte leído del puerto 40h.
IN	EAX, 20h;	Carga en EAX una doble word leída del puerto 20h.
IN	AX, DX;	Carga en AX una word leída del puerto contenido en DX.

- OUT: Escribe un byte, word o doble word de datos desde el registro especificado por el segundo operando (AL, AX, EAX) al puerto indicado por el primer operando. Si el número de puerto es de sólo 8 bits (0-255) se puede indicar el número literalmente, pero si se quiere acceder a cualquiera de los puertos (0-65535) se debe utilizar el registro DX para almacenar el número de puerto.

Ejemplos:

OUT	10h, AX;	Escribe la word contenida en AX en el puerto 10h.
OUT	DX, AL;	Escribe el byte presente en AL en el puerto indicado en DX.
OUT	DX, EAX;	Escribe la doble word de EAX en el puerto indicado en DX.

A partir de los microprocesadores 80286, Intel añadió la posibilidad de ejecutar transferencias de cadenas completas con una sola instrucción:

- INS: Transfiere datos desde un puerto especificado por el registro DX (segundo operando) a la zona de memoria apuntada por el registro índice destino (ES:DI, ES:EDI). INS no permite especificar un número de puerto literalmente, por lo que siempre se deberá cargar DX con el valor adecuado. Del mismo modo, se deberá preparar el registro índice destino (DI, EDI) con la dirección correcta antes de ejecutar la instrucción. Después de la transferencia, el registro índice destino avanzará o retrocederá automáticamente (dependiendo del estado del flag de dirección DF, que se modifica con las instrucciones CLD o STD), según el tamaño del dato transferido (1, 2 o 4 bytes). El primer operando sólo sirve para indicar el tamaño del dato a leer.

Ejemplos:

INS	AL, DX;	Lee un byte del puerto indicado por DX en ES:DI.
INS	AX, DX;	Lee una word del puerto indicado por DX en ES:DI.
INSW		Similar al anterior, sintaxis abreviada.

- OUTS: Transfiere datos desde un byte, word o doble word, indicado por el registro índice (SI, ESI) a un puerto señalado por el registro DX (primer operando). OUTS no permite especificar un número de puerto literalmente, por lo que siempre se deberá cargar DX con el valor adecuado. Del mismo modo, se deberá preparar el registro índice (SI, ESI) con la dirección correcta antes de ejecutar la instrucción. Después de la transferencia, el registro índice avanzará o retrocederá automáticamente (dependiendo del estado del flag de dirección DF), según el tamaño del dato transferido (1, 2 o 4 bytes). El segundo operando sólo sirve para indicar el tamaño de datos a transferir.

Ejemplos:

OUTS DX, AL;	Escribe el byte apuntado por SI en el puerto de DX.
OUTSB;	Idéntico al anterior, otra sintaxis.
OUTSD;	Escribe la doble word apuntada por SI en el puerto DX.

Los modernos micros contienen mecanismos de protección en el acceso a los puertos

Estas dos últimas instrucciones permiten el prefijo de repetición REP, para realizar entradas y salidas de bloques completos de datos, simplificando así la programación al no tener que utilizar bucles o registros intermedios, y acelerando la velocidad de transferencia.

La mayoría de los compiladores de C disponen de macros o pequeñas funciones para ofrecer la capacidad de acceso a los puertos, sin tener que introducir instrucciones in-line en ensamblador. Habitualmente se utilizan las siguientes funciones:

```
outp (unsigned numport, int value);
outpw (unsigned numport, unsigned value);
outport (int numport, int value);
outportb (int numport, unsigned char value);
int inp (unsigned numport);
unsigned inpw (unsigned numport);
int inpport (int numport);
unsigned char inportb (int numport);
```

Los modernos microprocesadores contienen dos mecanismos de protección para las entradas y salidas a través de los puertos, y sólo operan en modo protegido, incluyendo el modo virtual (V86):

- En el campo IOPL (I/O Priority Level) del registro EFLAGS, determina mediante dos bits el nivel de prioridad necesario para poder ejecutar instrucciones de entrada/salida. Las instrucciones dependientes del IOPL son: IN, INS, OUT, OUTS, CLI y STI.

- En el segmento TTS existe un campo de bits que permite o no las operaciones de entrada y salida. Por cada dirección de puerto, existe un bit que indica si es posible su acceso o no, y si alguno de los bits implicados (para acceder a un puerto de 32 bits, se comprueban 4 bits) está a 1, se generará una excepción. De esta forma, el sistema operativo puede otorgar derechos individuales de acceso a puertos a cada aplicación o tarea.

Existe otro sistema de direccionar a los periféricos, y es mapearlos directamente en memoria. Este sistema tiene la enorme ventaja de que es invisible para el procesador y se puede acceder a esta zona de memoria como si se tratara de cualquier otra. Por ejemplo, se puede transferir datos entre un registro y un periférico mediante la instrucción MOV, o con las instrucciones AND o TEST se pueden manipular los bits de los registros internos de un periférico. Un claro ejemplo son las modernas tarjetas con memoria interna direccionable que se conectan a los PC (tarjetas de vídeo o controladoras SCSI).

A continuación se ofrece una lista de los puertos estándar del PC, agrupados por conjuntos dentro del mismo periférico. Para cada puerto se indica su dirección, su tipo de acceso (si es de escritura e, de lectura l o de ambas cosas l/e), una breve descripción y algún comentario sobre su uso o sobre el tipo de ordenador en el que está aplicado. ■

DIRECCIONES DE LOS PUERTOS DE ENTRADA/SALIDA

0000-001F ——		DMA 1 (primer controlador 8237 de Acceso Directo a Memoria)
0000	l/e	DMA dirección canal 0, byte 0, después byte 1.
0001	l/e	DMA contador de words canal 0, byte 0, después byte 1.
0002	l/e	DMA dirección canal 1, byte 0, después byte 1.
0003	l/e	DMA contador de words canal 1, byte 0, después byte 1.
0004	l/e	DMA dirección canal 2, byte 0, después byte 1.
0005	l/e	DMA contador de words canal 2, byte 0, después byte 1.
0006	l/e	DMA dirección canal 3, byte 0, después byte 1.
0007	l/e	DMA contador de words canal 3, byte 0, después byte 1
0008	l	DMA registro de estado canales 0-3
	bit 7 = 1 petición canal 3	
	bit 6 = 1 petición canal 2	
	bit 5 = 1 petición canal 1	
	bit 4 = 1 petición canal 0	
	bit 3 = 1 contador terminal de canal para el canal 3	
	bit 2	= 1 contador terminal de canal para el canal 2
	bit 1	= 1 contador terminal de canal para el canal 1
	bit 0	= 1 contador terminal de canal para el canal 0
0008	e	DMA registro de comandos canales 0-3
	bit 7	= 1 sentido activo del DACK alto
		= 0 sentido activo del DACK bajo
	bit 6	= 1 sentido activo del DREQ alto
		= 0 sentido activo del DREQ bajo
	bit 5	= 1 selección escritura extendida
		= 0 selección escritura retardada
	bit 4	= 1 prioridad rotativa
		= 0 prioridad fija
	bit 3	= 1 temporizado (timing) comprimido
		= 0 temporizado (timing) normal
	bit 2	= 1 controlador activado
		= 0 transferencia de memoria a memoria activada
0009	e	DMA registro de petición de escritura
000A	l/e	DMA canal 0-3 registro de máscara
	bit 7-3	= 0 reservado
	bit 2	= 0 bit de reinicialización de máscara
		= 1 bit de activación de máscara
	bit 1-0 =	00 canal 0 seleccionado
		= 01 canal 1 seleccionado
		= 10 canal 2 seleccionado
		= 11 canal 3 seleccionado
000B	e	DMA canal 0-3 modo registro
	bit 7-6	= 00 modo demanda
		= 01 modo single
		= 10 modo bloque
		= 11 modo cascada
	bit 5	= 0 incremento de dirección seleccionado
		= 1 decremento de dirección seleccionado
	bit 3-2	= 00 operación de verificación
		= 01 escritura a memoria
		= 10 lectura de memoria
		= 11 reservado
	bit 1-0	= 00 canal 0 seleccionado
		= 01 canal 1 seleccionado
		= 10 canal 2 seleccionado
		= 11 canal 3 seleccionado
000C	e	DMA flip-flop inicilizador del puntero
000D	l	DMA registro temporal de lectura
000D	e	DMA inicializador maestro
000E	e	DMA registro de inicialización de máscara
000F	e	DMA registro de escritura de máscara
0010-001F ——		DMA controlador (8237) sobre los modelos PS/2 60 & 80
0020-003F ——		PIC 1 (Controlador Programable de Interrupcio- nes 8259)
0020	e	PIC palabra de inicialización de comandos ICW1
	bit 7-5	= 0 sólo utilizados en modo 80/85
	bit 4	= 1 ICW1 esta siendo utilizado
	bit 3	= 0 modo edge triggered
		= 1 modo level triggered
	bit 2	= 0 vectores de interrupción sucesivos utilizan 8 bytes
		= 1 vectores de interrupción sucesivos utilizan 4 bytes
	bit 1	= 0 modo cascada
		= 1 modo single, no es necesario el ICW3
	bit 0	= 0 no es necesario el ICW4
		= 1 es necesario el ICW4
0021	ePIC	ICW2,ICW3,ICW4 después ICW1 a 0020
		ICW2:
	bit 7-3 = líneas de direccionamiento A0-A3 de dirección base de vectores para el PIC	
	bit 2-0	= reservado
	ICW3:	
	bit 7-0	= 0 controlador esclavo no conectado al pin de interrupción correspondiente
		= 1 controlador esclavo conectado al pin de interrupción correspondiente
	ICW4:	
	bit 7-5	= 0 reservado
	bit 4	= 0 modo de no anidamiento-total
		= 1 modo especial de anidamiento-total
	bit 3-2	= 0x modo sin buffer
		= 10 modo/esclavo con buffer
		= 11 modo/maestro con buffer
	bit 1	= 0 normal EOI
		= 1 Auto EOI
	bit 0	= 0 modo 8085
		= 1 modo 8086/8088
0021	l/e	registro maestro de enmascaramiento de interrupciones del PIC
	OCW1:	
	bit 7 = 0 activación interrupción de parallel printer	
	bit 6 = 0 activación interrupción de disquete	
	bit 5 = 0 activación interrupción de disco duro	
	bit 4 = 0 activación interrupción de puerto serie 1	
	bit 3 = 0 activación interrupción de puerto serie 2	
	bit 2 = 0 activación interrupción de vídeo	
	bit 1 = 0 activación interrupción del teclado, ratón, RTC	
	bit 0 = 0 activación interrupción del temporizador	
	OCW2:	
	bit 7-5	= 000 rotar en el modo auto EOI (desactivado)
		= 001 sin EOI específica
		= 010 sin operación
		= 011 EOI especifica
		= 100 rotar en el modo auto EOI (activado)
		= 101 rotar sobre un comando EOI no específico
		= 110 activar prioridad de comando



= 111 rotar sobre un comando EOI específico
bit 4 = 0 reservado
bit 3 = 0 reservado
bit 2-0 petición de interrupción
0020 I Registros de petición de interrupción en servicio para OCW3
registro petición:
bit 7-0 = 0 no petición activa para la línea correspondiente de interrupciones
= 1 petición activa para la línea correspondiente de interrupciones
registro en-servicio:
bit 7-0 = 0 línea correspondiente no está actualmente en servicio
= 1 línea correspondiente está actualmente en servicio
0020 e PIC OCW3
bit 7 = 0 reservado
bit 6-5 = 0x no operación
= 10 reinicializa máscara especial
= 11 activa máscara especial
bit 4 = 0 reservado
bit 3 = 1 reservado
bit 2 = 0 no recogida de comandos
= 1 recogida de comandos
bit 1-0 = 0x no operación
= 10 lee el registro de petición de interrupciones en la siguiente lectura en 0020
= 11 lee el registro de interrupciones in-service en la siguiente lectura en 0020
0022-002B —— Intel 82355, parte del conjunto de chips para el 386sx
inicialización en modo POST desactivará estas direcciones, sólo un hard reset las reactivará de nuevo.
0040-005F —— PIT (Temporizador Programable de Interrupciones 8253, 8254)
XT & AT utilizan 40-43 PS/2 utiliza 40, 42,43,44, 47
0040 I/e PIT contador 0, contador divisor. (XT, AT, PS/2)
0041 I/e PIT contador 1, contador del refresco RAM, (XT, AT)
0042 I/e PIT contador 2, cassette & speaker, (XT, AT, PS/2)
0043 I/e PIT modo puerto, registro de control de palabras para contadores 0-2
bit 7-6 = 00 contador 0 seleccionado
= 01 contador 1 seleccionado (no en PS/2)
= 10 contador 2 seleccionado
bit 5-4 = 00 comando contador latch
= 01 contador lectura/escritura sólo bits 0-7
= 10 contador lectura/escritura sólo bits 8-15
= 11 contador lectura/escritura primero bits 0-7 , después 8-15
bit 3-1 = 000 modo 0 seleccionado
= 001 modo 1 seleccionado - programable de una sola vez
= x10 modo 2 seleccionado - generador de ratio
= x11 modo 3 seleccionado - generador de onda cuadrada
= 100 modo 4 seleccionado - estroboscopio disparado por software
= 101 modo 5 seleccionado - estroboscopio disparado por hardware
bit 0 = 0 contador binario 16 bits
= 1 BCD contador
0044 I/e PIT contador 3 (PS/2, EISA)
utilizado como temporizador de seguridad ante fallos, genera una NMI cuando el tiempo acaba.
Para NMI generadas por el usuario mirar en 0462.
0047 e Palabra de control del PIT registro contador 3 (PS/2, EISA)
bit 7-6 = 00 contador 3 seleccionado
= 01 reservado
= 10 reservado
= 11 reservado
bit 5-4 = 00 comando contador latch, contador 3
= 01 contador lectura/escritura, sólo bits 0-7
= 1x reservado
bit 3-0 = 00
0048 EISA
004A EISA
004B EISA
0060-006F —— Controlador teclado (8041, 8042) (o PPI (8255) sobre XT)
XT utiliza 60-63, AT utiliza 60-64
definiciones de los bits del puerto de entrada del controlador de teclado AT
bit 7 = 0 teclado inhibido
bit 6 = 0 CGA, otro valor MDA
bit 5 = 0 jumper fabricante instalado
bit 4 = 0 sistema de 512K de RAM, otro valor 640K
bit 3-0 reservado
AT teclado controlador input puerto bit definitions by Compaq
bit 7 = 0 llave de seguridad está bloqueada
bit 6 = 0 Pantalla Compaq dual-scan, 1=pantalla no Compaq
bit 5 = 0 Switch 5 de la placa del sistema esta a ON
bit 4 = 0 seleccionada auto velocidad, 1= seleccionada alta velocidad
bit 3 = 0 lento (4MHz), 1 = rápido (8MHz)
bit 2 = 0 80287 instalado, 1= NDP no instalado
bit 1-0 reservado
Definiciones de los bits del puerto de salida del controlador de teclado AT
bit 7 = salida de datos del teclado
bit 6 = salida del reloj del teclado
bit 5 = 0 buffer de entrada lleno
bit 4 = 0 buffer de salida vacío
bit 3 = reservado (ver nota)
bit 2 = reservado (ver nota)
bit 1 = puerta A20
bit 0 = reinicialización del sistema
Nota: Los bits 2 y 3 son el switch de la velocidad turbo o el bloqueo por palabra clave sobre las BIOS Award/AMI/Phoenix. Estos bits utilizan la funcionalidad BIOS no estándar del controlador de teclado para manipular:
pin 23 (8041 puerto 22) como interruptor turbo para AWARD
pin 35 (8041 puerto 15) como interruptor turbo para Phoenix
0060 I/e Puerto de datos del controlador del teclado o buffer de entrada del teclado (ISA, EISA)
Sólo puede ser leído cuando en el puerto de estado este el bit0 = 1
Sólo puede ser escrito cuando en el puerto de estado este el bit1 = 0
Comandos del teclado (los datos van también al puerto 0060):
ED dbl activa/desactiva el indicador Caps Num Scrl
EE sngl para diagnóstico, retorna EE.
EF-F2 sngl NOP (Sin operación). Reservado
F3 dbl Configurar la velocidad y el retardo del tecleo
F4 sngl teclado activado
F5 sngl teclado desactivado. Poner parámetros por defecto
F6 sngl cargar parámetros por defecto
F7-FD sngl NOP
FE sngl reenviar último scancode
FF sngl realizar la función interna de reinicialización power-on
0060 I Buffer de salida del controlador de teclado (vía PPI sobre XT)
0061 e Puerto B del controlador de teclado (ISA, EISA) (PS/2 puerto A está en 0092)
Puerto control del sistema para compatibilidad con 8255
bit 7 (1= reinicialización del IRQ 0)
bit 6-4 reservado
bit 3 = 1 chequeo de canal activado
bit 2 = 1 chequeo de paridad activado
bit 1 = 1 datos del altavoz activado
bit 0 = 1 Puerta del temporizador 2 al altavoz activado
0061 I registro de control del puerto B del controlador del teclado (ISA, EISA) puerto de control del sistema para compatibilidad con 8255
bit 7 chequeo de paridad occurrido
bit 6 chequeo de canal occurrido
bit 5 refleja el estado de la salida del temporizador 2
bit 4 se invierte con cada petición de refresco
bit 3 chequeo del estado del canal
bit 2 chequeo del estado de la paridad
bit 1 estado de los datos del altavoz
bit 0 estado de la puerta del temporizador 2 al altavoz
0064 I Estado del controlador del teclado (ISA, EISA)
bit 7 = 1 error de paridad en la transmisión desde el teclado
bit 6 = 1 timeout en recepción
bit 5 = 1 timeout en transmisión
bit 4 = 0 teclado inhibido
bit 3 = 1 datos en el registro de entrada es un comando
= 0 datos en el registro de entrada son datos
bit 2 = 0 estado del flag del sistema 0= reinicializado o reseteado 1=auto-testeo OK
bit 1 = 1 buffer de entrada lleno (las entradas 60/64 tienen datos para 8042)
bit 0 = 1 buffer de salida lleno (la salida 60 tiene datos para el sistema)
0064 e Buffer de entrada del controlador de teclado (ISA, EISA)
Comandos del controlador de teclado (los datos van al puerto 0060):
20 lee el byte cero de la RAM interna, este es el último comando de teclado enviado al 8041
21-3F lee el byte especificado en los 5 bits inferiores del comando en la RAM interna del 8041
60-7F dbl escribe el byte de datos a la dirección especificada en los 5 bits inferiores del comando.
Descripción del comando I/O 60 del teclado:
bit7 = 0, reservado
bit6 = modo compatibilidad IBM PC
bit5 = modo IBM PC
bit4 = desactivar teclado

bit3 = desactivar override
bit2 = flag sistema
bit1 = 0, reservado
bit0 = activo output buffer lleno interrupción
AB sngl Iniciar test del interface. Resultados:
0 = sinerror
1 = línea del reloj del teclado baja
2 = línea del reloj del teclado alta
3 = línea de datos del teclado baja
4 = línea de datos del teclado alta
AC lectura del volcado de diagnóstico. Los contenidos de la RAM del 8041, del puerto de salida, del puerto de entrada y la palabra de estado son enviados.
AD sngl desactiva el teclado (activando el bit 4 del byte de comando)
AE sngl activa teclado (limpiando el bit 4 del byte de comando)
AF AWARD Comando Mejorado: leer la versión del teclado
C0 leer puerto de entrada
C1 AWARD Comando Mejorado: conectar con el puerto de entrada, nibble bajo
C2 AWARD Comando Mejorado: conectar con el puerto de entrada, nibble alto
D0 leer puerto de salida
D1 dbl escribir puerto de salida. Próximo byte escrito en 0060 será escrito en el puerto de salida del 8041
D2 AWARD Comando Mejorado: escribir buffer de salida del teclado
D3 AWARD Comando Mejorado: escribir buffer de salida del dispositivo apuntador
D4 AWARD Comando Mejorado: escribir al dispositivo auxiliar
DD sngl desactivar la dirección line A20. Por defecto en modo real
DF sngl activar line A20
E0 read leer entradas de prueba. bit0 = T0 y bit1 = T1
Exx AWARD Comando Mejorado: activar puerto de salida
DE Compaq Esta es la parte segunda del comando para controlar el estado de los LEDs: NumLock, CapsLock y ScrollLock.
El byte segundo contiene el estado para configurar los LEDs.
bit 7-3 reservado. Debe ser 0.
bit 2 = 0 Caps Lock LED apagado
bit 1 = 0 Num Lock LED apagado
bit 0 = 0 Scroll Lock LED apagado
F0-FF sngl pulso del puerto de salida bajo por 6 microsegundos.
Los bits 0-3 contienen la máscara para los bits a ser pulsados.
Un bit es pulsado si su bit de máscara es cero.
Bit0=reseteo del sistema. ¡No poner a 0, sólo pulsarlo!
Nota general: Los controladores de teclado son muy diferentes unos de otros. Generalmente no se pueden intercambiar entre distintas máquinas.
Nota de Award: Extraído del documento Awardís Enhanced Keyboard Controller.
Nota de Compaq: Extraído de la Guía Técnica de Referencia Compaq Deskpro 386
0065 I Puerto de comunicaciones (Olivetti M24)
0070-007F —— CMOS RAM/RTC (Reloj en Tiempo Real MC146818)
0070 e registro puerto índice CMOS RAM (ISA, EISA)
bit 7 = 1 NMI desactivado
= 0 NMI activado
bit 6-0 índice CMOS RAM (64 bytes, algunas veces 128 bytes)
cualquier escritura a 0070 debe ser seguida por una acción a 0071 o el RTC quedará en un estado indefinido.
0071 I/e puerto de datos CMOS RAM (ISA, EISA)
registros RTC:
00 segundo actual en BCD
01 segundo de alarma en BCD
02 minuto actual en BCD
03 minuto de alarma en BCD
04 hora actual en BCD
05 hora de alarma en BCD
06 día de la semana en BCD
07 día del mes en BCD
08 mes en BCD
09 año en BCD (00-99)
0A registro de estado A
bit 7 = 1 actualización en progreso
bit 6-4 divisor que identifica la frecuencia en la que se basa el reloj
bit 3-0 velocidad seleccionada para la frecuencia de salida y velocidad de interrupción.
0B registro de estado B
bit 7 = 0 ejecutar
= 1 parar
bit 6 = 1 interrupción periódica activa
bit 5 = 1 interrupción de alarma activa
bit 4 = 1 interrupción update-ended activa
bit 3 = 1 interrupción de onda cuadrada activa
bit 2 = 1 el calendario está en formato binario
= 0 el calendario está en formato BCD
bit 1 = 1 modo 24-horas
= 0 modo 12-horas
bit 0 = 1 Hora adelantada activada. Sólo en USA. Sin uso en Europa. Algunas versiones del DOS limpian este bit cuando se utiliza el comando DATE/TIME.
0C registro de estado C
bit 7 = flag de petición de interrupción
bit 6 = flag interrupción periódica
bit 5 = flag interrupción de alarma
bit 4 = flag de actualización de interrupción
bit 3-0 flag reservado
0D registro de estado D
bit 7 = 1 El reloj de tiempo real tiene energía
bit 6-0 reservado
0E byte de diagnóstico del estado
bit 7 = 0 RTC pierde energía
bit 6 = 1 checksum del CMOS RAM erróneo
bit 5 = 1 información de la configuración inválida en el POST
bit 4 = 1 tamaño de la memoria erróneo en el POST
bit 3 = 1 inicialización fallida del disco duro/adaptador
bit 2 = 1 tiempo del CMOS RAM encontrado inválido
bit 1 = 1 configuración de los adaptadores no coincide (EISA)
bit 0 = 1 time out leyendo in ID de un adaptador (EISA)
0F byte de estado de apagado shutdown
00 = ejecución normal del POST
01 = inicialización del chip para reentrada en modo real
04 = salto al código bootstrap
05 = genera un EOI en el salto al puntero Dword en 40:67
06 = salto a la Dword en 40:67 sin EOI
07 = retorno a la INT15/87 (mover bloque)
08 = retorno al test de memoria POST
09 = retorno a la INT15/87 (mover bloque)
0A = salto al puntero Dword en 40:67 sin EOI
0B = retorna IRETS a través de 40:67
10 Tipo de la unidad de disco para A: y B:
bit 7-4 tipo de unidad de la unidad 0
bit 3-0 tipo de unidad de la unidad 1
= 0000 sin unidad
= 0001 360K
= 0010 1M2
= 0011 720K
= 0100 1M44
= 0101-1111 reservado
11 reservado / Configuración Extendida AMI CMOS (AMI Hi-Flex BIOS)
bit 7 = 1 Programación de la velocidad del teclado
bit 6-5 = 00 Retardo de la velocidad del teclado 250 mSec
bit 4-0 = 00011 Retardo del teclado 21.8 Chars/Sec
12 tipo de disco duro para la unidad 0 y la unidad 1
bit 7-4 tipo de unidad para la unidad 0
bit 3-0 tipo de unidad para la unidad 1
si cualquiera de los nibbles es igual a 0F, entonces los bytes19 y 1A son validos.
13 reservado / Configuración Extendida AMI CMOS (AMI Hi-Flex BIOS)
bit 7 = 1 Opción de soporte de ratón
bit 6 = 1 Test de la memoria sobre 1 MB desactivado
bit 5 = 1 Sonido del test de memoria desactivado
bit 4 = 1 Chequeo de error de paridad de la memoria activado
bit 3 = 1 Mostrado del mensaje Hit <ESC> desactivado
bit 2 = 1 Area de datos del disco duro tipo 47 en la dirección 0:300
bit 1 = 1 Espera de la tecla <F1> si existe algún error activado
bit 0 = 1 Estado de Num Lock es ON al arrancar
14 byte de equipamiento
bit 7-6 unidades de disco instaladas
= 00 1 disquetera instalada
= 01 2 disqueteras instaladas
= 10 reservado
= 11 reservado
bit 5-4 Pantalla principal
= 00 Tarjeta adaptadora con opción ROM
= 01 40*25 color
= 10 80*25 color
= 11 monocromo
bit 3-2 reservado
bit 1 = 1 coprocesador instalado (no Weitek)
bit 0 disquetera disponible en el arranque
15 LSB de memoria base en Kb
16 MSB de memoria base en Kb
17 LSB de memoria extendida total en Kb
18 MSB de memoria extendida total en Kb
19 unidad C byte de extensión
1A unidad D byte de extensión
1B-27 reservado
1B/1C word para el registro de comparación 82335 RC1 en [24] (Phoenix)

1D/1E word para el registro de comparación 82335 RC2 en [26]
(Phoenix)
29-2D reservado
29/2A word para el registro de comparación Intel 82335 CC0 en
[28](Phoenix)
2B/2C word enviada al registro de comparación 82335 CC1 en [2A]
(Phoenix)
2D Configuración Extendida AMI CMOS (AMI Hi-Flex BIOS)
(la BIOS Phoenix chequea los valores AA o CC)
bit 7 = 1 Procesador Weitek no presente
bit 6 = 1 Floppy Unidad Seek At Boot desactivado
bit 5 = 1 Secuencia de arranque C:, A:
bit 4 = 1 Velocidad de arranque es alta
bit 3 = 1 Memoria Cache activada
bit 2 = 1 Memoria Interna Cache <1>
bit 1-0 reservado
2E CMOS MSB checksum sobre 10-2D
2F CMOS LSB checksum sobre 10-2D
30 LSB de memoria extendida sobre 1Mb en POST
31 MSB de memoria extendida sobre 1Mb en POST
32 siglo en BCD
33 flags de información
bit4 = bit4 del registro CR0 de la CPU (Phoenix)
este bit también es conocido como RESERVADO INTEL
34-3F reservado
34 bit4 bit5 (Phoenix BIOS)
3D/3E word para el registro de configuración de memoria 82335 MCR
en [22](Phoenix)
3D bit3 tamaño de memoria base 512/640 (Phoenix)
3E bit7 = 1 recolocación activada (Phoenix)
bit1 = 1 video shadow activada (Phoenix)
bit0 = 1 shadow BIOS activada (Phoenix)
Parámetros de Unidad Definibles por el Usuario son también
almacenados en RAM CMOS:
AMI (386sx BIOS 1989) primera unidad definible por el usuario (tipo 47)
1B L cilindros
1C H cilindros
1D cabezas
1E L Precompensación en la escritura al cilindro
1F H Precompensación en la escritura al cilindro
21 L Zona de aparque de los cilindros
22 H Zona de aparque de los cilindros
23 sectores
AMI (386sx BIOS 1989) segunda unidad definible por el usuario(tipo 48)
24 L cilindros
25 H cilindros
26 cabezas
27 L Precompensación en la escritura al cilindro
28 H Precompensación en la escritura al cilindro
2A L Zona de aparque de los cilindros
2B H Zona de aparque de los cilindros
2C sectores
Phoenix (386BIOS v1.10.03 1988) primera unidad definible por el
usuario (tipo 48)
20 L cilindros
21 H cilindros
22 cabezas
23 L Precompensación en la escritura al cilindro
24 H Precompensación en la escritura al cilindro
25 L Zona de aparque de los cilindros
26 H Zona de aparque de los cilindros
27 sectores
Phoenix (386BIOS v1.10.03 1988) segunda unidad definible por el
usuario (tipo 49)
(cuando la opción de palabra clave del PS/2 no es utilizada)
35 L cilindros
36 H cilindros
37 cabezas
38 L Precompensación en la escritura al cilindro
39 H Precompensación en la escritura al cilindro
3A L Zona de aparque de los cilindros
3B H Zona de aparque de los cilindros
3C sectores
0080-008F ——Registros de página DMA (74612)
0080 I/e registro extra de página (almacenamiento temporal)
0081 I/e DMA canal 2 address byte 2
0082 I/e DMA canal 3 address byte 2
0083 I/e DMA canal 1 address byte 2
0084 I/e registro extra de página
0085 I/e registro extra de página
0086 I/e registro extra de página
0087 I/e DMA canal 0 address byte 2
0088 I/e registro extra de página
0089 I/e DMA canal 6 address byte 2
008A I/e DMA canal 7 address byte 2
008B I/e DMA canal 5 address byte 2
008C I/e registro extra de página
008D I/e registro extra de página
008E I/e registro extra de página
008F I/e registro refresco de página DMA
00A0-00AF —— PIC 2 (Controlador Programable de Interrupciones 8259)
00A0 I/e PIC 2 igual que 0020 para el PIC 1
00A1 I/e PIC 2 igual que 0021 para el PIC 1 excepto para OCW1:
bit 7 = 0 reservado
bit 6 = 0 interrupción disco duro activada
bit 5 = 0 interrupción excepción del coprocesador activada
bit 4 = 0 interrupción del ratón activada
bit 3 = 0 reservado
bit 2 = 0 reservado
bit 1 = 0 cascada redireccionada activada
bit 0 = 0 interrupción del reloj en tiempo real activada
00C0 —— TI SN746496 generador programable de tonos,PCjr
00C0-00DF—— DMA 2 (segundo controlador 8237 (Direct Memory Access controlador))
00C0 I/e DMA canal 4 dirección de memoria bytes 1 y 0 (bajo) (ISA, EISA)
00C2 I/e DMA canal 4 contador de la transferencia bytes 1 y 0 (bajo) (ISA, EISA)
00C4 I/e DMA canal 5 dirección de memoria bytes 1 y 0 (bajo) (ISA, EISA)
00C6 I/e DMA canal 5 contador de la transferencia bytes 1 y 0 (bajo) (ISA, EISA)
00C8 I/e DMA canal 6 dirección de memoria bytes 1 y 0 (bajo) (ISA, EISA)
00CA I/e DMA canal 6 contador de la transferencia bytes 1 y 0 (bajo) (ISA, EISA)
00CC I/e DMA canal 7 dirección de memoria bytes 1 y 0 (bajo) (ISA, EISA)
00CE I/e DMA canal 7 contador de la transferencia bytes 1 y 0 (bajo). (ISA, EISA)
00D0 I DMA canal 4-7 registro de estado (ISA, EISA)
bit 7 = 1 petición canal 7
bit 6 = 1 petición canal 6
bit 5 = 1 petición canal 5
bit 4 = 1 petición canal 4
bit 3 = 1 contador terminal para el canal 7
bit 2 = 1 contador terminal para el canal 6
bit 1 = 1 contador terminal para el canal 5
bit 0 = 1 contador terminal para el canal 4
00D0 e DMA canal 4-7 registro de comandos (ISA, EISA)
bit 7 = 1 sentido activo del DACK alto
= 0 sentido activo del DACK bajo
bit 6 = 1 sentido activo del DREQ alto
= 0 sentido activo del DREQ bajo
bit 5 = 1 selección escritura extendida
= 0 selección escritura retardada
bit 4 = 1 prioridad rotativa
= 0 prioridad fija
bit 3 = 1 temporizado (timing) comprimido
= 0 temporizado (timing) normal
bit 2 = 0 controlador activado
bit 1 = 1 transferencia de memoria a memoria activada
bit 0
00D2 e DMA canal 4-7 registro del petición de escritura (ISA, EISA)
00D4 e DMA canal 4-7 registro máscara de escritura única (ISA, EISA)
bit 7-3 reservado
bit 2 = 0 limpiar bit de máscara
= 1 activar bit de máscara
bit 1-0 = 00 canal 4 seleccionado
= 01 canal 5 seleccionado
= 10 canal 6 seleccionado
= 11 canal 7 seleccionado
00D6 e DMA canal 4-7 modo registro (ISA, EISA)
bit 7-6 = 00 modo demanda
= 01 modo single
= 10 modo bloque
= 11 modo cascada
bit 5 = 0 incremento dirección seleccionado
= 1 decremento dirección seleccionado
bit 4 = 0 auto-inicialización desactivada
= 1 auto-inicialización activada
bit 3-2 = 00 operación de verificación
= 01 escribir a memoria
= 10 leer desde memoria
= 11 reservado
bit 1-0 = 00 canal 4 seleccionado
= 01 canal 5 seleccionado
= 10 canal 6 seleccionado
= 11 canal 7 seleccionado
00D8 e DMA canal 4-7 flip-flop inicilizador del puntero (ISA, EISA)
00DA I DMA canal 4-7 registro temporal de lectura (ISA, EISA)
00DA e DMA canal 4-7 master clear (ISA, EISA)
00DC e DMA canal 4-7 registro de inicialización de máscara (ISA, EISA)
00DE e DMA canal 4-7 registro máscara de escritura (ISA, EISA)
00E0 registro de partición de direcciones, sólo para registros de
codificación de memoria PS/2m80
00F0-00FF —— Coprocesador (8087..80387)

0100-0107 —— PS/2 POS (Selector Programable de Opciones)
0170-0177 —— HDC 2 (Controlador 2/ de Disco duro) igual que 01Fx (ISA, EISA)
01F0-01F7 —— HDC 1 (Controlador 1/de Disco duro) igual que 017x (ISA, EISA)
01F0 I/e registro de datos
01F1 I registro de error
errores de diagnóstico de modo:
bit 7-3 reservado
bit 2-1 = 001 no detectado error
= 010 error de formateo de dispositivo
= 011 error del buffer de sector
= 100 error de circuitería ECC
= 101 error de control del microprocesador
modo de operación:
bit 7 = 1 bloque defectuoso detectado
= 0 bloque OK
bit 6 = 1 error ECC irrecuperable
= 0 no error
bit 5 reservado
bit 4 = 1 ID encontrado
= 0 ID no encontrado
bit 3 reservado
bit 2 = 1 comando completado
= 0 comando abortado
bit 1 = 1 pista 000 no encontrada
= 0 pista 000 encontrada
bit 0 = 1 DAM no encontrada
= 0 DAM encontrada (CP-3022 siempre a 0)
01F1 e WPC/4 (Write Precompensation Cylinder dividido por 4)
01F2 I/e contador de sector
01F3 I/e numero de sector
01F4 I/e cilindro bajo
01F5 I/e cilindro alto
01F6 I/e unidad/cabeza
bit 7 = 1
bit 6 = 0
bit 5 = 1
bit 4 = 0 unidad 0 seleccionada
= 1 unidad 1 seleccionada
bit 3-0 bits de cabeza seleccionada
01F7 I registro de estado
bit 7 = 1 el controlador está ejecutando un comando
bit 6 = 1 la unidad está preparada
bit 5 = 1 fallo escritura
bit 4 = 1 búsqueda completada
bit 3 = 1 buffer de sector requiere
01F7 I registro de estado
bit 7 = 1 el controlador está ejecutando un comando
bit 6 = 1 la unidad está preparada
bit 5 = 1 fallo escritura
bit 4 = 1 búsqueda completada
bit 3 = 1 buffer de sector requiere *asistencia*
bit 2 = 1 datos de disco leidos correctamente corregidos
bit 1 = 1 índice - puesto a 1 cada vuelta del disco
bit 0 = 1 comando previo acabado en un error
01F7 e registro de comandos
comandos:
98 E5 modo chequeo de energía (IDE)
90 ejecutar diagnóstico de la unidad
50 formatear pista
EC identificar unidad (IDE)
97 E3 desocupado (IDE)
95 E1 desocupado (IDE)
91 inicializar parámetros de unidad
1x recalibrar
E4 leer buffer (IDE)
C8 leer DMA con reintento (IDE)
C9 leer DMA sin reintento(IDE)
C4 leer múltiple (IDE)
20 leer sectores con reintento
21 leer sectores sin reintento
22 leer *long* con reintento
23 leer *long* sin reintento
40 leer sectores de verificación con reintento
41 leer sectores de verificación sin reintento
7x buscar
EF activar características (IDE)
C6 activar modo múltiple (IDE)
99 E6 activar modo *sleep* (IDE)
96 E2 espera (IDE)
94 E0 espera inmediatamente (IDE)
E8 escribir buffer (IDE)
CA escribir DMA con reintento (IDE)
CB escribir DMA sin reintento (IDE)

C5 escribir múltiple (IDE)
E9 escribir lo mismo (IDE)
30 escribir sectores con reintento
31 escribir sectores sin reintento
32 escribir *long* con reintento
33 escribir *long* sin reintento
3C escribir verificación (IDE)
9A único vendedor (IDE)
C0-C3 único vendedor (IDE)
8x único vendedor (IDE)
F0-F4 EATA estándar (IDE)
F5-FF único vendedor(IDE)
0200-020F —— Espacio I/O reservado para el Puerto de Juegos
0200-0207 —— Puerto de Juegos, ocho direcciones idénticas en algunas placas
0201 I leer posición y estado del joystick
bit 7 estado B joystick botón 2 / D botón paddle
bit 6 estado B joystick botón 1 / C botón paddle
bit 5 estado A joystick botón 2 / B botón paddle
bit 4 estado A joystick botón 1 / A botón paddle
bit 3 B coordenada Y joystick / D coordenada paddle
bit 2 B coordenada X joystick / C coordenada paddle
bit 1 A coordenada Y joystick / B coordenada paddle
bit 0 A coordenada X joystick / A coordenada paddle
e fuego joysticks de cuatro disparo-único
0278-027E —— Puerto paralelo impresora, igual que 0378 y 03BC
0278 e puerto de datos
0279 I/e puerto de estado
027A I/e puerto de control
02A2-02A3 —— Reloj MSM58321RS
02B0-02DF —— EGA alternativa, EGA primaria en 03C0
02C0-02Cx —— Reloj AST
02E0-02EF —— Adquisición de datos (AT)
02E2 Adquisición datos (adaptador 0)
02E3 Adquisición datos (adaptador 0)
02E8-02EF —— puerto serie, igual que 02F8, 03E8, and 03F8
02F8-02FF —— puerto serie, igual que 02E8, 02F8, and 03F8
02F8 e registro de mantenimiento de transmisión
02F8 I registro del buffer del receptor
I/e divisor *latch*, byte bajo, cuando DLAB=1
02F9 I/e divisor *latch*, byte alto, cuando DLAB=1
I/e registro de activación de interrupción cuando DLAB=0
02FA I registro identificación de interrupciones
02FB I/e registro control línea
02FC I/e registro control módem
02FD I registro de estado de línea
02FF I/e registro *scratch*
0320-0323 —— XT HDC 1 (Controlador Disco Duro)
0348-0357 —— DCA 3278
0360-036F —— PC network (AT)
0360-0367 —— PC network (sólo XT)
0370-0377 —— FDC 2 (Controlador 2 Floppy Disk 8272) igual que 03F0
0372 e controlador de disquete DOR (Registro Digital de Salida)
0374 I controlador de disquete registro de estado
0375 I/e controlador de disquete registro de datos
0376 I/e controlador de disco duro registro de datos
0377 I controlador de disquete DIR (Registro Digital de Entrada)
0377 e seleccionado registro para la velocidad de transferencia de datos
0378-037A —— Puerto paralelo impresora, igual que 0278 y 03BC
0378 e puerto de datos
0379 I/e puerto de estado
037A I/e puerto de control
0380-038F —— Adaptador 2 de Control de Unión de Datos Binarios Síncronos (ver 03A0)
0380 I/e sobre la placa 8255 puerto A, sentido interno/externo
0381 I/e sobre la placa 8255 puerto B, interface módem externo
0382 I/e sobre la placa 8255 puerto C, control interno y *gating*
0383 I/e sobre la placa 8255 registro de modo
0384 I/e sobre la placa 8253 canal generador onda cuadrada
0385 I/e sobre la placa 8253 canal 1 *time-out* inactividad
0386 I/e sobre la placa 8253 canal 2 *time-out* inactividad
0387 I/e sobre la placa 8253 registro de modo
0388 I/e sobre la placa 8273 lectura: estado escritura: comando
0389 I/e sobre la placa 8273 lectura: respuesta escritura: parámetro
038A I/e sobre la placa 8273 estado de la interrupción transmisión
038B I/e sobre la placa 8273 estado de la interrupción de recepción
038C I/e sobre la placa 8273 dato
0390-039F —— Adaptador Cluster (AT)
0390-0393 (adaptador 0) (XT)
03B0-03BF —— MDA (Adaptador Pantalla Monocroma basado en 6845)
03B0 igual que 03B4
03B1 igual que 03B5
03B2 igual que 03B4
03B3 igual que 03B5

03B4	e	MDA CRT registro índice (EGA/VGA)
		seleccionado con el registro (0-11h) es para ser accedido a través de
3B5		
03B5	l/e	MDA CRT registro datos (EGA/VGA)
		seleccionado por el puerto 3B4. Registros C-F pueden ser leídos
		00 total horizontal
		01 visualizado horizontal
		02 posición sincronismo horizontal
		03 anchura de pulso de sincronismo horizontal
		04 total vertical
		05 visualizado vertical
		06 posición sincronismo vertical
		07 anchura de pulso de sincronismo vertical
		08 modo entrelazado
		09 máximo de líneas horizontales
		0A inicio cursor
		0B fin cursor
		0C dirección inicio alta
		0D dirección inicio baja
		0E posición cursor alta
		0F posición cursor baja
		10 lápiz óptico alto
		11 lápiz óptico bajo
03B6		igual que 03B4
03B7		igual que 03B5
03B8	l/e	MDA registro de control de modo
		bit 7 no utilizado
		bit 6 no utilizado
		bit 5 parpadeo activo
		bit 4 no utilizado
		bit 3 video activo
		bit 2 no utilizado
		bit 1 no utilizado
		bit 0 modo alta resolución
03B9		reservado para el registro de selección de color en un adaptador de color
03BA	l CRT registro de estado	EGA/VGA: registro de estado de entrada 1
	bit 7-4	reservado
	bit 3	video blanco/negro
	bit 2-1	reservado
	bit 0	controlador horizontal
	bit 7	(MSD dice) si este bit cambia después de un lectura 8000h, entonces:
	bit 6-4	000 = adaptador es Hercules o compatible
		001 = adaptador es Hercules+
		101 = adaptador es Color Hercules
		si no: adaptador es desconocido
03BA	e	EGA/VGA registro de control de características
03BB		reservado para la reinicialización del estroboscopio del lápiz óptico
03BC-03BF	——	Puerto paralelo impresora, igual que 0278 y 0378
03BC	e	puerto de datos
03BD	l/e	puerto de estado
	bit 7 = 0	ocupado
	bit 6 = 0	reconocimiento
	bit 5 = 1	sin papel
	bit 4 = 1	impresora es seleccionada
	bit 3 = 0	error
	bit 2 = 0	una IRQ ha ocurrido
	bit 1-0	reservado
03BE	l/e	puerto de control
	bit 7-5	reservado
	bit 4 = 1	IRQ activa
	bit 3 = 1	seleccionar impresora
	bit 2 = 0	inicializar impresora
	bit 1 = 1	desplazamiento de línea automático
	bit 0 = 1	estroboscopio
03BF	l/e	Registro de configuración Hercules
03C0-03CF	——	EGA (Primer Adaptador Gráfico Mejorado) alternativa a 02C0
03C0	(l)/e	EGA VGA ATC registro índice/datos
03C1	l VGA	otro registro de atributos
03C2	l EGA VGA	registro de entrada de estado 0
	e VGA	registro de salida variado
03C3	l/e VGA	subsistema de video activo
03C4	e EGA	TS registro índice
	l/e VGA	registro índice secuenciado
03C5	e EGA	TS registro de datos
	l/e VGA	otro registro de secuenciado
03C6	l/e VGA	PEL registro máscara
03C7	l/e VGA	PEL modo de lectura de dirección
	l VGA	DAC registro de estado
03C8	l/e VGA	PEL modo escritura de dirección
03C9	l/e VGA	PEL registro de datos
03CA	e EGA	registro de posición gráficos 2
	l VGA	registro de control de características
03CC	e EGA	registro de posición gráficos 1

	l VGA	registro de salida variado
03CE	e EGA	GDC registro índice
	l/e VGA	registro de dirección de gráficos
03CF	e EGA	GDC registro de datos
	l/e VGA	otro registro gráfico
03D0-03DF	——	CGA (Adaptador Gráfico de Color)
03E8-03EF	——	puerto serie, igual que 02E8, 02F8, and 03F8
03F0-03F7	——	FDC 1 (1st Floppy Disk Controlador 8272) igual que 0370
03F0		estado A del controlador disquete (PS/2)
	bit 7	interrupción pendiente
	bit 6	unidad segunda instalada
	bit 5	paso
	bit 4	pista 0
	bit 3	cabeza 1 seleccionado
	bit 2	índice
	bit 1	protegido contra escritura
	bit 0	dirección
03F1	l	estado B del controlador disquete (PS/2)
	bit 7-6 reservado	
	bit 5	unidad seleccionada (0=A:, 1=B:)
	bit 4	escribir datos
	bit 3	leer datos
	bit 2	escritura activa
	bit 1	motor activo 1
	bit 0	motor activo 0
03F2	e	DOR del controlador disquete (Registro de Salida Digital)
	bit 7-6	reservado PS/2
	bit 7 = 1	unidad 3 motor activo
	bit 6 = 1	unidad 2 motor activo
	bit 5 = 1	unidad 1 motor activo
	bit 4 = 1	unidad 0 motor activo
	bit 3 = 1	DMA del disquete activo (reservado PS/2)
	bit 2 = 1	FDC activo (reinicio del controlador)
		= 0 mantener FDC en el reinicio
	bit 1-0	unidad seleccionada (0=A 1=B ..)
03F4	l	registro de estado del controlador de disquete
	bit 7 = 1	el registro de datos esta preparada
	bit 6 = 1	la transferencia es desde el controlador al sistema
	= 0	la transferencia es desde el sistema al controlador
	bit 5 = 1	modo no-DMA
	bit 4 = 1	controlador disquete ocupado
	bit 3 = 1	unidad 3 ocupada (reservado en el PS/2)
	bit 2 = 1	unidad 2 ocupada (reservado en el PS/2)
	bit 1 = 1	unidad 1 ocupada
	bit 0 = 1	unidad 0 ocupada
03F5	l	registro de comando de disquete, estado 0
	bit 7-6	último comando de estado
		= 00 comando terminado satisfactoriamente
		= 01 comando terminado anormalmente
		= 10 comando inválido
		= 11 terminado anormalmente por cambio en la señal *ready*
	bit 5	= 1 búsqueda completada
	bit 4	= 1 chequeo del equipamiento ocurrido después de un error
	bit 3	= 1 no preparada
	bit 2	= 1 número de cabeza en la interrupción
	bit 1-0 = 1	unidad seleccionada (0=A 1=B ..)
		(sobre PS/2 01=A 10=B)
		registro de estado 1
	bit 7	final de cilindro; número de sector mayor que sectores/pista
	bit 6 = 0	
	bit 5 = 1	error de CRC en ID o campo de datos
	bit 4 = 1	*overrun*
	bit 3 = 0	
	bit 2 = 1	ID de sector no encontrado
	bit 1 = 1	protección contra escritura encontrada durante la escritura
	bit 0 = 1	marca de ID de dirección no encontrada
		registro de estado 2
	bit 7 = 0	
	bit 6 = 1	marca de dirección de datos borrada encontrada
	bit 5 = 1	error de CRC en datos
	bit 4 = 1	cilindro erróneo detectado
	bit 3 = 1	comando *scan* satisfactorio
	bit 2 = 1	comando *scan* fallada, sector no encontrado
	bit 1 = 1	cilindro defectuoso, ID no encontrado
	bit 0 = 1	marca de dirección de datos desaparecida
		registro de estado 3
	bit 7	señal de estado de fallo
	bit 6	estado de protección contra escritura
	bit 5	estado preparada
	bit 4	estado pista cero
	bit 3	señal de estado de doble cara
	bit 2	cara seleccionada (cabeza seleccionada)
	bit 1-0	unidad seleccionada (0=A 1=B ..)
03F6	l/e	registro de datos del controlador de disco duro

bit 7-4 reservado
bit 3 = 0 reduce escritura actual
1 cabeza seleccionada
bit 2 = 1 inicialización del disco activada
0 inicialización del disco desactivada
bit 1 = 0 inicialización del disco activada
1 inicialización del disco desactivada
bit 0 reservado

03F7 l controlador disquete DIR (Registro Entrada Digital)
bit 7 = 1 cambio diskette
bit 6 disco duro puerta de escritura
bit 5 disco duro cabeza seleccionado 3
bit 4 disco duro cabeza seleccionado 2
bit 3 disco duro cabeza seleccionado 1
bit 2 disco duro cabeza seleccionado 0
bit 1 disco duro unidad 1 seleccionado
bit 0 disco duro unidad 0 seleccionado, tiene conflictos con:
bit 0 diskette alta densidad seleccionado

03F7 e registro selección para la velocidad de transferencia de datos del diskette
bit 7-2 reservado
bit 1-0 = 00 modo 500 Kb/S
= 01 modo 300 Kb/S
= 10 modo 250 Kb/S
= 11 reservado

03F8-03FF —— puerto serie (8250,8251,16450,16550), igual que 02E8,02F8,and 03F8

03F8 e puerto serie, registro de mantenimiento de transmisión, que contiene los caracteres a ser enviados. El bit 0 es enviado primero.
bit 7-0 bits de datos, cuando DLAB=0 (Divisor Latch Access Bit)
l registro del buffer de recepción, contiene el carácter recibido.
El bit 0 es enviado primero.
bit 7-0 bits de datos cuando DLAB=0 (Divisor Latch Access Bit)
l/e byte bajo *divisor latch* cuando DLAB=1

03F9 l/e byte alto *divisor latch* cuando DLAB=1
l/e registro de interrupción activa cuando DLAB=0
bits 7-4 reservado
bit 3 = 1 interrupción del estado del módem activa
bit 2 = 1 interrupción del estado de la línea de recepción activa
bit 1 = 1 interrupción de registro de mantenimiento de la transmisión vacío activa
bit 0 = 1 interrupción de datos en la recepción disponibles activa (y 16550 *timeout*)
- 16550 generará una interrupción si existen datos en el FIFO y no han sido leídos dentro del tiempo que tarda en recibir cuatro bytes o si ningún dato es recibido dentro del tiempo que tarde en recibir cuatro bytes.

03FA l registro de identificación de interrupción. Información sobre una interrupción pendiente es almacenada aquí. Cuando el registro es direccionado,la interrupción con prioridad más alta es mantenida, y ninguna otra interrupción es reconocida hasta que la CPU atienda a esa interrupción.
bit 7-6 = 00 reservados sobre 8250, 8251, 16450
= 11 si las colas FIFO están activas (sólo 16550)
bit 5-4 = 0 reservados
bit 3 = 0 reservado en 8250, 16450
= 1 interrupción pendiente 16550
bit 2-1 identifican la interrupción pendiente con la mayor prioridad
= 11 interrupción del estado de la línea de recepción. Prioridad = la más alta
= 10 interrupción de datos disponibles en el registro de recepción. Prioridad = segunda
= 01 interrupción de registro de transmisión vacío. Prioridad = tercera
= 00 interrupción del estado del módem. Prioridad = cuarta
bit 0 = 0 interrupción pendiente. El contenido de este registro puede ser usado como un puntero a la rutina apropiada de manejo de interrupciones.
1 sin interrupción pendiente
- los flags de interrupción pendiente usan lógica inversa, 0=pendiente, 1=no
- las interrupciones ocurrirán si alguno de los bit del estado de línea están activados

03FA e 16650 FCR (Registro de Control del FIFO)
bit 7-6 = 00 1 byte
= 01 4 bytes
= 10 8 bytes
= 11 14 bytes
bit 5-4 = 00 reservado
bit 3 = 1 cambia los pins RXRDY TXRDY desde el modo 0 al modo 1
bit 2 = 1 limpia el FIFO XMIT
bit 1 = 1 limpia el FIFO RCVR
bit 0 = 1 limpia las colas FIFO XMIT y RCVR
- el bit 0 debe ser activado antes de escribir a los otros bits FCR

03FB l/e registro de control de línea
bit 7 = 1 bit de acceso al divisor latch (DLAB)
0 registro de acceso al buffer de recepción, mantenimiento de transmisión, o interrupción activa.
bit 6 = modo *break* activado.
bit 5 = paridad *stick*
bit 4 = 1 paridad par seleccionada
bit 3 = paridad activada
1 número para de unos son enviados y chequeados en los bits de las palabras de datos y en el bit de paridad.
0 número impar de unos son enviados y chequeados
bit 2 = 0 bit uno-parar
1 bit cero-parar
bit 1-0 00 longitud de palabra es de 5 bits
01 longitud de palabra es de 6 bits
10 longitud de palabra es de 7 bits
11 longitud de palabra es de 8 bits

03FC l/e registro de control del módem
bit 7-5 = 0 reservado
bit 4 = 1 modo *loopback* para comprobación de la salida por el puerto serie. En este modo los datos transmitidos son recibidos inmediatamente así que la CPU puede verificar las salidas y accesos a las transmisiones de datos por el puerto serie.
bit 3 = 1 salida auxiliar designada por el usuario 2
bit 2 = 1 salida auxiliar designada por el usuario 1
bit 1 = 1 activa la petición para enviar obligada
bit 0 = 1 activa el DTR obligado

03FD l registro de estado de línea
bit 7 = 0 reservado
bit 6 = 1 registros de mantenimiento y desplazamiento de la transmisión vacíos
bit 5 = 1 registro de mantenimiento de la transmisión vacío. El controlador esta preparado para aceptar nuevos caracteres a enviar.
bit 4 = 1 interrupción *break*. El dato de entrada recibido es mantenido en el estado de bit a cero más tiempo que el tiempo de iniciar bit + bits de datos + bit de paridad + bits de paro.
bit 3 = 1 error. El bit de paro que sigue a los últimos bits de paridad o de datos está a 0.
bit 2 = 1 error de paridad. El carácter tiene una paridad errónea.
bit 1 = 1 error. Un carácter fue enviado al buffer receptor antes que el carácter previo del buffer fuera leído. Esto destruye el carácter previo.
bit 0 = 1 dato disponible. Un carácter de llegada ha sido recibido y mandado al registro del buffer de entrada.

03FE l registro de estado de módem
bit 7 = 1 detectado carrier de datos
bit 6 = 1 indicador de llamada
bit 5 = 1 datos preparados
bit 4 = 1 limpiar para enviar
bit 3 = 1 detectado *carrier* de datos *delta*
bit 2 = 1 indicador del sobrante del timbre de llamada
bit 1 = 1 datos preparados *delta*
bit 0 = 1 limpiar para enviar *delta*
- bits 0-3 son reinicializados cuando la CPU lee el MSR
- bit 4 es el Registro de Control del Módem RTS durante el testeo loopback
- bit 5 es el Registro de Control del Módem DTR durante el testeo loopback
- bit 6 es el Registro de Control del Módem OUT1 durante el testeo loopback
- bit 7 es el Registro de Control del Módem OUT2 durante el testeo loopback

03FF l/e registro *scratch*
¡Las direcciones sobre 03FF sólo se aplican a las máquinas EISA!
1000-1FFF slot 1 EISA
2000-2FFF slot 2 EISA
3000-3FFF slot 3 EISA
4000-4FFF slot 4 EISA
5000-5FFF slot 5 EISA
6000-6FFF slot 6 EISA
7000-7FFF slot 7 EISA

06E2-06E3 —— adquisición de datos (adaptador 1)
0790-0793 —— cluster (adaptador 1)
0800-08FF —— registros de los puertos de E/S para CMOS extendidas RAM o SRAM
0AE2-0AE3 —— cluster (adaptador 2)
0B90-0B93 —— cluster (adaptador 2)
0C00 l/e registro página para escribir a SRAM o I/O
0C80-0C83 —— registros ID placa del sistema
1390-1393 —— cluster (adaptador 3)
2390-2393 —— cluster (adaptador 4)
3220-3227 —— puerto serie 3, descripción igual que 03F8
3228-322F —— puerto serie 4, descripción igual que 03F8

Gracias a Wim Osterholt por su estupenda recopilación sobre los puertos I/O

EL MODO DE LAS VENTANAS

Bernardo García

Después de haber dedicado un tiempo a los controles básicos de Windows, y sin olvidarnos de ellos, es el momento de conocer más profundamente los diferentes tipos de ventanas que se pueden manejar, y algunas técnicas y herramientas para facilitar la creación de programas.

De entre las posibles clasificaciones se ha elegido esta por ser, quizás, la mas gráfica; nos referimos a las ventanas Modales (Modal) y No Modales (Modeless). Ambos términos, que carecen de una apropiada traducción al español, se refieren al comportamiento de unas ventanas con otras cuando están activadas.

MODAL Y NO MODAL

Para mostrar la diferencia (no siempre clara) entre ambos términos se han introducido en el programa de ejemplo varios tipos de ventanas. La primera división que se puede realizar es entre ventanas y diálogos. Aunque el término diálogo se refiere originalmente a

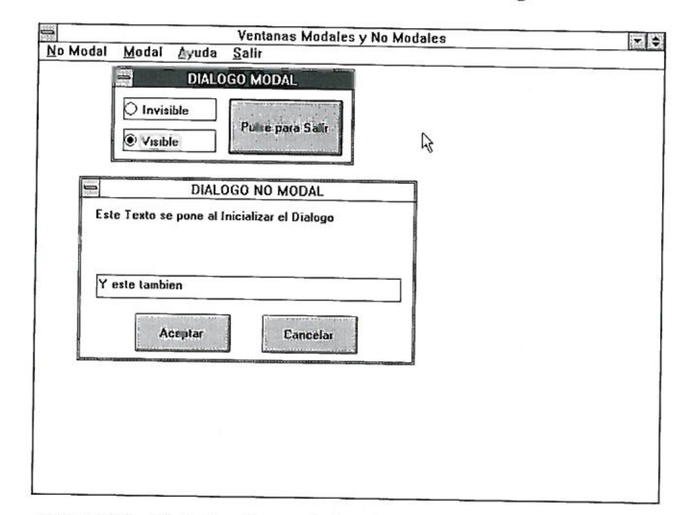


FIGURA 1. Distintos tipos de bordes en ventanas.

la función que desempeña, es decir, comunicarse con el usuario para mostrar u obtener información, poseen casi siempre la característica de ser ventanas modales. Este tipo de ventana no permite que se realice otro tipo de operaciones mientras no se les haya prestado la debida atención, como suele ser el caso de las cajas de mensajes o el clásico Acerca De. Para indicar su condición de diálogo, este tipo de ventanas muestra un borde diferente, como puede verse en la figura 1.

En esta característica radica la diferencia entre Modal y No Modal. Mientras que la primera requiere la atención del usuario, la segunda le permite cambiar a otra ventana sin necesidad de ser cerrada.

Para comprender más claramente las diferencias entre ambos tipos se puede ejecutar el programa ejemplo. En el menú de la aplicación aparecen cuatro opciones, la primera de las cuales muestra una ventana No Modal. Se puede observar que su aspecto es similar al de cualquier otra, siendo posible seleccionar la ventana principal de la aplicación para escoger otra opción. Si se pulsa Modal aparece otra ventana, cuyo aspecto es también similar; sin embargo, una vez abierta es imposible volver al menú.

Una ventana modal requiere la atención del usuario

Esta segunda ventana sirve para mostrar que la diferencia entre ambos tipos no es más que funcional y nada tiene que ver con su aspecto. Si desea cerrar la ventana deberá hacer visible el botón de salir, que aparecerá al seleccionar el botón de radio descrito como "Visible", como se ve en la figura 2.

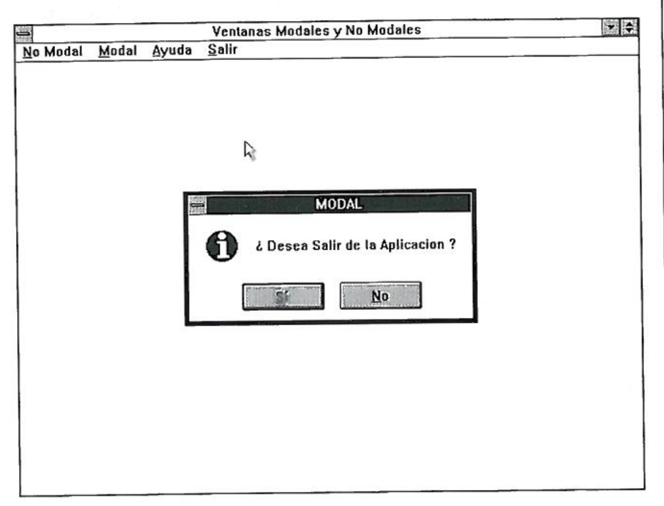


FIGURA 2. Ejemplo de ventanas modales y no modales.

En el programa aparecen tres ventanas modales; ésta es la primera de ellas. Además está la ventana de Acerca De que, junto con la de Salir, pertenecen todas al mismo tipo, siendo diferente la forma de crearlas, como veremos mas adelante.

EDITOR DE RECURSOS

Como se ha mencionado en artículos anteriores, para crear los recursos el método más sencillo es utilizar un Editor de Recursos. Cada compilador suele traer el suyo propio, no existiendo un estándar. A pesar de ello, es indiferente cuál se utilice, ya que los ficheros de recursos emplean un lenguaje estándar, lo que permite utilizarlos con cualquier compilador. Por esta razón se ha preferido no entrar en los detalles de un editor de recursos en particular, sino tratar estos de forma genérica.

En la figura 3 se pueden ver los recursos de que dispone la aplicación. En primer lugar se encuentra el menú principal, MenuModal, con las opciones e identificadores de las mismas.

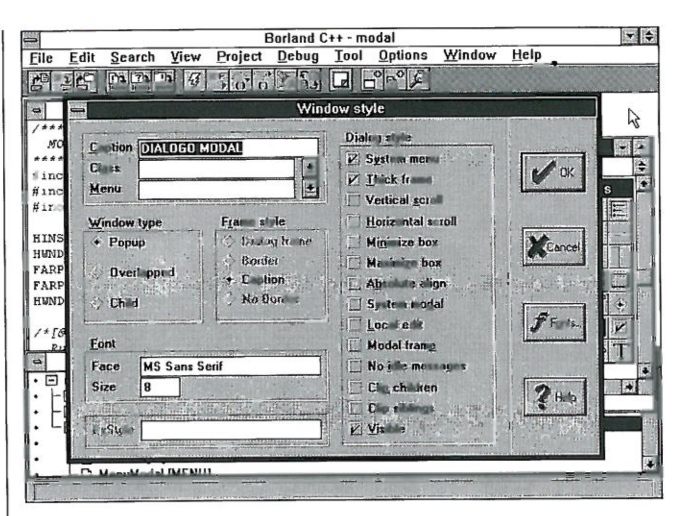


FIGURA 3. Recursos de la aplicación.

Como es habitual, se encuentra el dialogo DLG_ACERCA, que se corresponde con la ayuda. Los siguientes recursos son DLG_NOMODAL y DLG_MODAL que, como sus nombres indican, corresponden a cada uno de estos tipos de ventanas. En la figura 4 se puede ver el texto generado por el editor para ambos diálogos.

Al crear los diálogos, una de las opciones que se pueden especificar es el estado de los controles. En DLG_MODAL se puede observar que uno de ellos tiene la propiedad NOT WS_VISIBLE, indicando que el control en cuestión no estará visible al llamar al diálogo desde la aplicación.

Cuando la aplicación se activa, ShowWindow() maximiza la ventana

Si se comparan ambos diálogos no parecen tener más diferencia que los controles que se han colocado en cada uno, y efectivamente así es. Para comprender qué es lo que les hace comportarse de formas tan distintas se debe analizar el código del programa.

PROGRAMA DE EJEMPLO

El proceso de inicialización permanece casi sin modificaciones, registrando y creando la ventana principal de la aplicación. Pero a continuación se pueden empezar a observar los cambios. Las siguientes líneas de código:

```
lpprocNOMODAL = MakeProcInstance((FARPROC)ProcNoModal, hInstance);
lpprocMODAL = MakeProcInstance((FARPROC)ProcModal, hInstance);
```

nos van a permitir asociar las funciones de tratamiento

de mensajes declaradas al principio del fichero con la instancia actual del programa, de forma similar a lo que se viene haciendo al llamar al diálogo de Acerca De.

Otro cambio que se puede observar es la modificación del bucle de lectura de mensajes. Anteriormente los mensajes eran tratados por una sola función, ya fueran destinados a la ventana principal o a los controles en ella situados. En este caso es necesario diferenciar si los mensajes pertenecen o no a la ventana principal, utilizando para ello la función IsDialogMessage().

CreateDialog() crea un recurso especificado por el usuario

MainWndProc sigue encargándose del tratamiento de parte de los mensajes, en particular de los dirigidos a la aplicación. Para que la aplicación aparezca maximizada cada vez que se active se ha tratado el mensaje WM_ACTIVATE, llamando a ShowWindow() cuando la aplicación se activa para mostrar la ventana en su máximo tamaño. WM_PAINT es otro mensaje relacionado con la estética del programa. Aunque las órdenes BeginPaint() y EndPaint() no parezcan tener mucho sentido en solitario, obligan a repintar toda la ventana.

En el mensaje WM_COMMAND es donde se realiza la

FIGURA 4

```
DLG_NOMODAL DIALOG 5, 15, 194, 100
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | WS_THICK-
FRAME
CAPTION "DIALOGO NO MODAL"
FONT 8, "MS Sans Serif"
{
 DEFPUSHBUTTON "Aceptar", IDOK, 32, 72, 56, 24
 PUSHBUTTON "Cancelar", IDCANCEL, 106, 72, 56, 24
 LTEXT "Text", IDC_TEXTO, 8, 6, 178, 37
 EDITTEXT IDC_EDIT, 9, 48, 178, 15
}
DLG_MODAL DIALOG 6, 14, 138, 45
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | WS_THICK-
FRAME
CAPTION "DIALOGO MODAL"
FONT 8, "MS Sans Serif"
{
 DEFPUSHBUTTON "Pulse para Salir", IDOK, 67, 6, 66, 35, BS_DEFPUSHBUTTON |
NOT WS_VISIBLE | WS_TABSTOP
 CONTROL "Invisible", IDC_RADIOBUTTON1, "BUTTON", BS_AUTORADIOBUTTON |
WS_BORDER, 5, 6, 54, 14
 CONTROL "Visible", IDC_RADIOBUTTON2, "BUTTON", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_BORDER, 5, 26, 54, 14
}
```

primera parte del trabajo de este ejemplo: la creación de las ventanas. En último lugar aparece IDM_ACERCA, uno de los casos de ventana Modal. La función MakeProcInstance() asocia la función AcercaDe() con la aplicación, tras lo cual se llama a DialogBox() para mostrar una ventana Modal, y espera a que ésta se cierre para continuar la ejecución del programa. Otro caso parecido ocurre con MessageBox(), que muestra una caja de diálogo con un texto y botones, utilizada habitualmente para mostrar avisos.

```
rc = MessageBox(hWnd, "¿ Desea Salir de la Aplicación ?",
        "MODAL", MB_ICONASTERISK|MB_YESNO);
```

Esta línea mostrará una caja preguntando al usuario si desea salir. Los valores MB_ICONASTERISK y MB_YESNO indican el aspecto que debe presentar la ventana y el número y tipo de botones.

Estos diálogos, al ser propios del sistema, presentan un borde alrededor de la ventana que los identifica como de tipo modal.

El tratamiento de IDM_NO_MODAL e IDM_MODAL es idéntico. En ambos se comprueba que la ventana no esté abierta, y en ese caso se llama a CreateDialog() para crearla. La principal característica de esta función es que utiliza un diálogo creado por el usuario con el editor de recursos. En un ejemplo anterior se utilizó CreateWindow() para crear cada uno de los controles contenidos en un diálogo. En esta ocasión sólo es necesario llamar a una función para obtener el mismo resultado. Además, no es necesario conocer las coordenadas exactas de los controles: el editor nos permite diseñar gráficamente el aspecto del diálogo (ver figura 5) y calcula las coordenadas de los elementos empleados.

Una buena herramienta simplifica enormemente la tarea de diseño

La siguiente llamada a CreateDialog() realiza todo el proceso de creación de los controles del diálogo DLG_NOMODAL.

```
DialogoNoModal = CreateDialog(hInst,
MAKEINTRESOURCE(DLG_NOMODAL),hWnd,lpprocNOMODAL);
```

Esta función realiza llamadas a otras funciones, entre ellas CreateWindow(), para obtener el mismo resultado

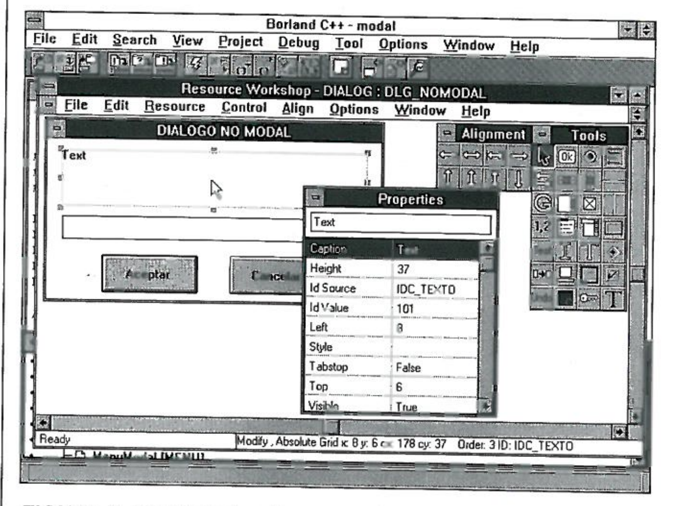


FIGURA 5. Propiedades de una ventana.

que si el programa fuera creando, uno por uno, todos los controles que componen el diálogo. De esta forma, con sólo una llamada a una función se simplifica el proceso de crear una ventana que dependa de la ventana principal.

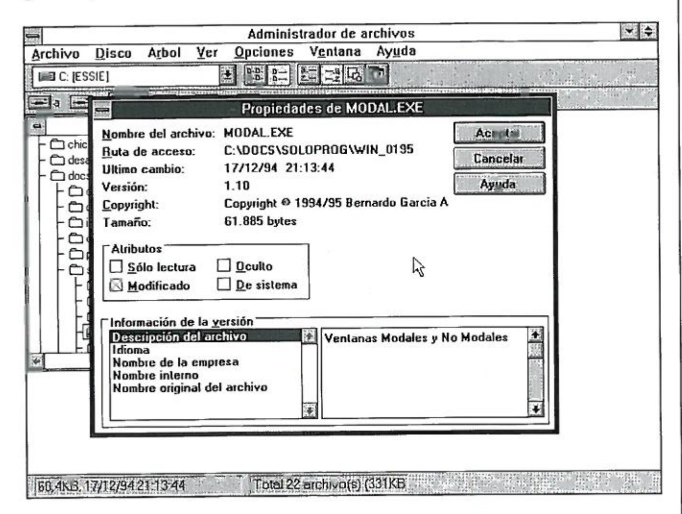


FIGURA 6. Propiedades de una ventana.

Además de los parámetros que identifican la instancia de la aplicación (hInst) y la ventana principal de la misma (hWnd), tenemos lpprocNOMODAL. Esta resulta ser una de las variables que se utilizaron al principio del programa para asociar la rutina de tratamiento de mensajes del diálogo. Tanto ésta como lpprocMODAL contienen la dirección de la función que debe tratar los mensajes de cada uno de estos diálogos. Es precisamente en estas rutinas donde se encuentra la diferencia entre la ventana modal y la no modal de la aplicación.

lpprocNOMODAL contiene la dirección de ProcNoModal(). Esta función no realiza ningún proceso en especial. Trata el mensaje WM_INITDIALOG empleado para inicializar el diálogo y rellenar los controles que lo componen. El proceso escogido resulta más bien simple, consistiendo en rellenar los controles de la ventana con algunos textos.

Se ha añadido un nuevo recurso: la información de versión

Además de los identificadores creados por el usuario, como IDM_SALIR, existen otros predefinidos de Windows, como son IDOK e IDCANCEL. El primero se produce al pulsar el botón de OK en un diálogo, y el otro al pulsar el botón de cancelar. Al recibir cualquiera de estos valores se cierra el diálogo, limpiando DialogoNoModal para permitir que se pueda abrir de nuevo.

Un proceso similar ocurre con lpprocMODAL, que apunta a ProcModal(). Al inicializar el diálogo se activa uno de los botones con CheckRadioButton(), y a continuación se realizan los procesos necesarios para convertir la ventana en Modal. La característica principal de este tipo de diálogos es la imposibilidad de acceder a las demás ventanas, bien de la aplicación o de Windows en el caso de ser System Modal. Para simular este comportamiento se desactivan las otras ventanas con EnableWindow().

Existen identificadores predefinidos, como IDOK e IDCANCEL

En este caso es necesario comprobar si el diálogo no modal estaba abierto. En el caso opuesto no es necesario, ya que resulta imposible entrar en el diálogo no modal desde éste.

Una vez desactivadas las ventanas anteriores resulta imposible salir del diálogo modal. La única manera de terminar es pulsar el botón de salir, que está oculto. Para hacerlo visible se debe pulsar el botón de radio correspondiente, y lo mismo si se quiere ocultar de nuevo. Para realizar esta operación se emplea la orden:

```
ShowWindow(GetDlgItem(hDlg, IDOK), TRUE);
```

que está compuesta por una llamada a GetDlgItem() (que nos devuelve el identificador del control) y otra a ShowWindow() para mostrar u ocultar ese control.

Si se pulsa el botón de salir, dependiendo de las ventanas que hubiera abiertas antes de entrar en el diálogo modal, se debe volver a la última de ellas que estuviera abierta (la principal o la ventana no modal), utilizando para ello la función SetFocus().

Para salir del programa hay varias posibilidades: a través del menú propio de la ventana o con la opción de Salir del menú principal de la misma. En esta última se mostrará el mensaje de aviso que se ha visto anteriormente, y caso de que se pulse el botón de OK se enviará el mensaje para terminar la aplicación.

Al terminar la ejecución del programa se llama a FreeProcInstance() para destruir la asociación realizada al principio de la aplicación entre las funciones de tratamiento de las ventanas y la instancia actual del programa. Sin embargo, en este caso esta operación no es necesaria, ya que al terminar la aplicación, tanto ésta como las funciones son destruidas. A pesar de ello, es conveniente acostumbrarse a liberar todos los recursos asignados durante la ejecución de la aplicación, ya que otras aplicaciones pueden necesitarlos después.

OPCIONES DE LOS DIALOGOS

En el momento de diseñar un diálogo, el editor de recursos dispone de más propiedades de las que se han empleado aquí. En el diseño de las ventanas se ha

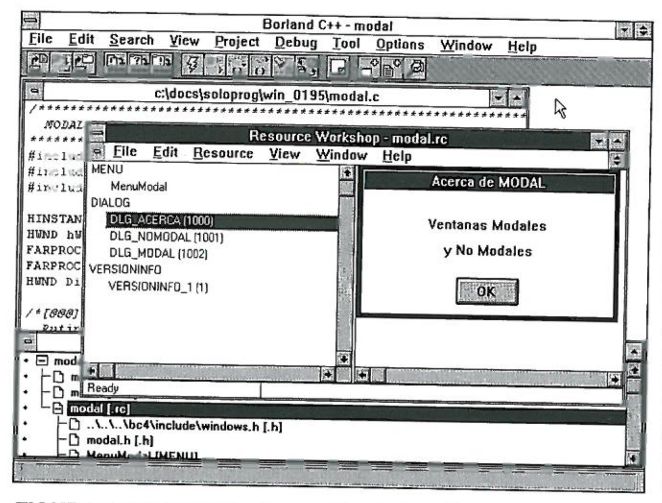


FIGURA 7. Información de versión de la aplicación.

evitado emplear varias de estas propiedades. Se han empleado bordes de ventana normales, identificados por la propiedad Thick Frame (ver figura 6), cuando se podía haber empleado Modal Frame para dar el aspecto adecuado a cada diálogo. Otro de los posibles valores es System Modal, que identifica a la ventana como modal, pero respecto a todo el sistema en vez de a la ventana madre.

MAS RECURSOS

Además de todos los empleados hasta el momento, también se ha añadido un nuevo recurso al programa ejemplo: la información de versión. Aunque en este caso no se utiliza por la propia aplicación, estos datos de identificación pueden ser consultados por cualquier programa. Como ejemplo, acceda al administrador de archivos y, tras seleccionar el fichero ejecutable, pulse ALT+ENTER para visualizar esta información. En la figura 7 se puede ver el resultado obtenido con esta operación.

CONCLUSION

Con un editor de recursos es relativamente sencillo modificar el fichero de recursos para cambiar el aspecto de un diálogo. La apariencia externa de la aplicación es uno de los puntos que más tiempo puede necesitar a la hora del diseño. Una buena herramienta, en este caso un editor de recursos, puede simplificar enormemente la tarea en comparación con su alternativa: realizar el proceso a mano.

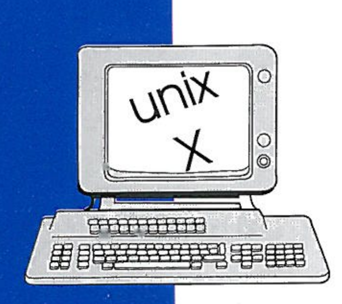
El diseño del interfaz de usuario en una aplicación, requiere gran esfuerzo

Se ha empezado a ver las diferentes propiedades de las ventanas, con algo aparentemente tan sencillo, y a la vez tan importante, como su comportamiento social. Sin entrar en otras características mucho más complejas, se puede observar la cantidad de posibilidades que ofrecen las ventanas, tanto modales como no modales.

Uniendo la sencillez de diseño que permite un editor de recursos con los diferentes tipos de ventanas, se puede crear en poco tiempo un interface de usuario muy vistoso. Si a esto se le añade el código necesario para que la aplicación no sea una simple fachada, el proceso completo de desarrollo se simplifica enormemente. ■

1os MOVIMIENTOS EN UNIX (y II)

Fernando J. Echevarrieta



En el capítulo anterior se dieron los primeros pasos por el sistema, primero mirando alrededor y más tarde echando a andar. En este capítulo, se abordarán algunos conceptos que permitan desenvolverse ya con cierta soltura, aumentando la potencia de los comandos vistos a través de redirecciones y control de procesos concurrentes. También se presentará la estructura de permisos de UNIX y los distintos tipos o grupos de usuario.

Continuamos este mes la andadura que desde hace algunos números estamos dedicando a este gran sistema operativo que es UNIX. Este mes se descubrirán nuevos lugares por los que encaminarse.

REDIRECCIONES Y PIPES

En UNIX la forma de realizar redirecciones es fuertemente dependiente de la shell utilizada; sin embargo, para estos primeros pasos se pueden destacar las características comunes a todas ellas que son, a la vez, las más utilizadas. En principio, se podría señalar que cada proceso UNIX tiene asociadas 3 vías de entrada/salida (E/S) estándares:

- Una salida estándar (STDOUT), que por defecto se encuentra conectada al terminal (pantalla) asociado al proceso;
- Una entrada estándar (STDIN), que por defecto se encuentra asociada al mismo terminal (en este caso, al teclado);
- Una salida de error (STDERR), que por defecto se encuentra conectada con la salida SD.

Estos puntos E/S pueden ser redirigidos de una manera en apariencia similar al DOS. Así, escribiendo > nombre_fichero tras el nombre de un comando, producirá la redirección de su salida a un fichero. De manera análoga se actuará con la entrada, escribiendo < nombre_fichero. También es posible realizar redirecciones de la salida de error, que se estudiarán en el tema dedicado a shells. El fichero LSETC.TXT, que se encuentra en el disco, ha sido generado mediante la instrucción:

```
ls -lF /etc > lsetc.txt
```

Este ejemplo ilustra también la opción -F del comando ls, que hará que aparezca un carácter / junto al nombre de los directorios, un * junto al nombre de los ejecutables y un @ junto a los enlaces simbólicos. El comando ls en linux, además, mostrará de distinto color cada tipo de fichero, según se indique en el /etc/DIR_COLORS.

En el capítulo anterior se veía como el comando cat podía mostrar el contenido de un fichero: cat fichero.

Del mismo modo, se podría haber redirigido la salida para copiar el fichero: cat fichero > fichero. Otro ejemplo de utilización sería no indicar el parámetro de entrada, con lo que el comando leería de la STDIN (teclado). Esto es algo que se puede realizar con un tipo especial de comandos UNIX llamados filtros que se estudiará más adelante. Así, por ejemplo: cat > notas copiaría todo lo que se tecleara al fichero notas (para indicar al comando que se ha finalizado habría que teclear CTRL + D). Equivaldría a un copy con: notas en DOS.

Una forma muy potente de comunicación entre procesos es el llamado pipe. Como se señalaba en el capítulo de introducción, un usuario puede ejecutar varios procesos e interconectarlos a través de pipes o tuberías, simbolizados por el carácter | (ASCII 124). A través de un pipe se realiza una llamada al sistema que interconecta la STDOUT de un comando con la STDIN del siguiente. Ambos procesos se ejecutarán concurrentemente, es decir, en PARALELO. No hay que confundir un pipe UNIX con el mecanismo del DOS que utiliza el mismo símbolo, donde no existe concurrencia de procesos sino paso de información a través de ficheros temporales. Como ejemplo útil, se podría teclear ls | sort | more , que produciría una salida en pantalla y paginada del contenido del directorio actual ordenado alfabéticamente.

MANEJO DE IMPRESORAS

En estos primeros pasos, se supondrá(n) instalada(s) la(s) impresora(s) y se comentará su manejo a nivel de usuario. Desde esta perspectiva, el manejo de impresoras se realizará a través de los siguientes comandos:

lpr -P<impresora> fichero: Envía el fichero mencionado a la impresora deseada.

lpq -P<impresora>: Muestra el estado de la impresora, así como los trabajos en la cola de impresión y sus respectivos propietarios.

lprm: Borra trabajos de la cola de impresión.

Para usuarios deseosos de profundizar más en este momento se recomienda la consulta del manual en línea de estos comandos, así como de las páginas lpd, lpc y printcap (una buena práctica sería buscar también con man -k printer).

PROTECCION DE ARCHIVOS Y CONTROL DE ACCESO

Al ser UNIX un sistema multiusuario surge el problema de la protección y privacidad de la información. Así, cada fichero posee un código de 9 bits para regular su acceso. El esquema empleado, clásico en muchos sistemas operativos, consiste en dividir el universo de usuarios que ve cada fichero en tres clases: la clase u (user), formada únicamente por el dueño del fichero; la clase g (group), formada por todos los usuarios que pertenecen al mismo grupo del dueño y la clase o, formada por el resto del universo (o como diría Gonzalo León, "el universo se divide en yo, mis amigos y mis enemigos").

Un usuario puede pertenecer a más de un grupo pero un fichero sólo puede pertenecer a uno. De esta forma, parte de los ficheros de un usuario podrían ser accedidos por uno de los grupos a los que el usuario pertenece y parte por otro grupo. Para entender mejor las relaciones de inclusión entre grupos, consúltese la figura 1.

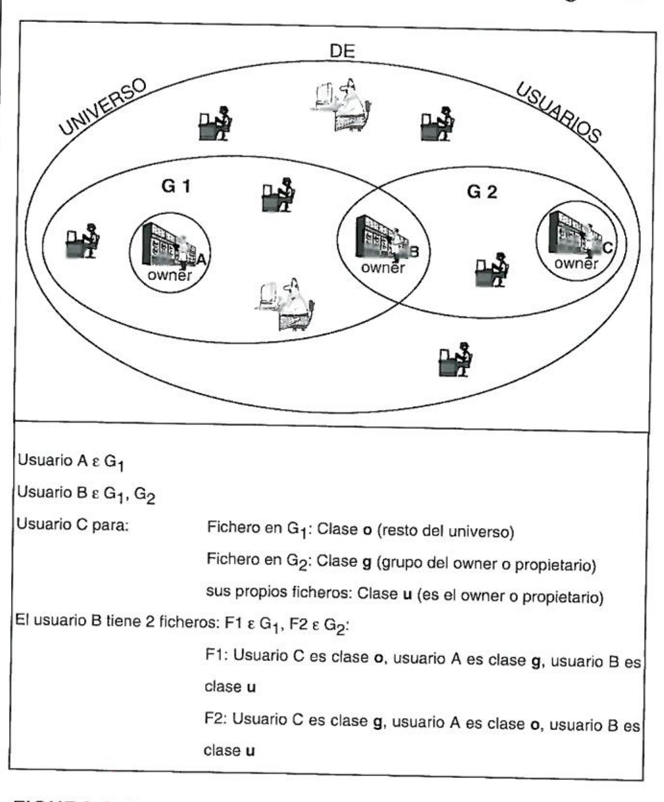


FIGURA 1: Estructura del universo de usuarios UNIX.

Como siempre, es el root el que decide qué usuarios pertenecen a qué grupos, los cuales se suelen organizar atendiendo a razones de trabajo. La lógica de esto es que un usuario concreto (clase u), puede tener en un fichero una carta de su novia, que no le interesa que lea nadie más. Sin embargo, también dispondrá de una serie de ficheros a los que tendrá que permitir el acceso a su grupo de trabajo (clase g) pero no querrá que los vea nadie más. Del mismo modo, podría también interesarle que todo el mundo (clase o) pudiera acceder a la información contenida en otra serie de ficheros. El root, como superusuario, es caso aparte, ya que dispone de acceso a todos los ficheros del sistema.

Existen 3 formas de acceder a un fichero: lectura, escritura y ejecución. Así, los 9 bits de protección de acceso de cada fichero se encuentran divididos en 3 grupos de 3 bits. Cada grupo de 3 bits indica acceso a u, g, o, respectivamente y cada bit de cada grupo indica:

- bit 1 (r), permiso de lectura;
- bit 2 (w), permiso de escritura;
- bit 3 (x), permiso de ejecución.

EXAMEN DE CADENAS DE PERMISO

No hay mejor forma de entender el proceso que acceder a las cadenas de bits de protección de cada fichero, primero para su lectura y más tarde para su modificación. Para ello, se empleará la opción -l del comando ls que proporciona una salida como:

—rwxr-xr-x 1 echeva 127 Jan 20 1:24 fichero

La salida aquí presentada se obtendría de un UNIX "estándar". En Linux, se obtendrá la presentada en el fichero LSECT.TXT. Observando la figura 2 se entenderá mejor el significado de los bits de la cadena. Cuando un bit está activo, se mostrará la letra correspondiente a la forma de acceso que representa y si está inactivo aparecerá como un guión. Así, el ejemplo presenta un fichero con permisos de lectura, escritura y ejecución (rwx) para el propietario, y de sólo lectura y ejecución (r-x) para todos los demás. En algunos casos puede aparecer una s en el lugar de un bit de permiso, que indicará un setuid o setgid, es decir, herencia de propiedades de grupo o de propietario para el usuario que ejecute el fichero.

En el capítulo anterior se mencionaba que un proceso adquiría los permisos del sujeto que lo había lanzado. Sin embargo, en ocasiones, puede interesar que se hereden los permisos del propietario del fichero. Es el caso del comando passwd, proceso que debe poder ejecutar cualquiera y en cambio, tener permisos de root para modificar el fichero /etc/passwd que sólo tiene permiso de escritura para el mismo root. Examinando sus cadenas de permiso en el archivo LSECT.TXT se puede ver cómo dispone de factor de herencia. Se puede también observar que existe un décimo carácter que definirá el tipo de fichero:

- s, si es un socket;
- l, si es un enlace simbólico;
- d si es un directorio,
- c si es un dispositivo de caracteres;
- b, si es un dispositivo de bloque;
- -, si es un fichero corriente (se recuerda que para UNIX todo son ficheros).

Para saber a qué grupo pertenece cada fichero se puede utilizar ls -lg. En Linux la opción -g carece de significado, ya que queda englobada directamente en la opción -l. En general, existe un grupo principal al que pertenece cada usuario, cuyo número se encuentra en su entrada en el fichero /etc/passwd. Es a este grupo al que pertenecerán por defecto todos los ficheros de ese usuario.

CAMBIO DE LAS CADENAS DE PERMISO

Para alterar los permisos de un fichero, UNIX provee la instrucción chmod. Este comando reconoce dos tipos de sintaxis: una basada en cadenas mnemónicas expresando modificaciones (más intuitiva) y otra basada en combinaciones en base octal (más potente). La sintaxis para el primer caso es:

chmod quién op permiso [[op2 permiso2]...] fichero

donde quién expresa el conjunto de usuarios a los que va a afectar el cambio de permisos, y puede ser una combinación cualquiera de u, g, o ó la letra a, o ninguna letra para designar a todos. op representa la operación a realizar y puede ser + para añadir, - para quitar o = para "resetear" el permiso. Y permiso, obviamente, representa el permiso que se desea modificar. Así, por ejemplo, chmod g -r fichero, quitará el permiso de lectura para la clase g (grupo) y dejará el resto de los permisos inalterados; chmod +x fichero hará ejecutable un fichero para todo el universo; y chmod ug +x fichero lo hará sólo para el usuario y su grupo.

La segunda sintaxis del comando, aunque más compleja, permite cambiar de una sola vez todos los permisos de un fichero: si se trata por separado cada uno de los grupos de usuario, se encontrará 3 bits de permiso para cada uno. Con 3 bits podrán expresarse hasta 8 combinaciones, desde 000 hasta 111. Así, se puede expresar con un dígito octal los permisos de cada grupo. Por poner un ejemplo: supongamos que se desea que todo el universo pueda leer un fichero pero sólo el propietario pueda modificarlo. La estructura de permisos será: u (rw), g (r), o (r). Es decir rw- r— r— o lo que es lo mismo 110 100 100 que en octal será 644. Por tanto, habrá que teclear chmod 644 fichero.

CONTROL DE MULTIPLES TRABAJOS

El hecho tantas veces repetido de que UNIX sea un S.O. multiusuario y multitarea implica necesariamente la concurrencia de procesos. Se denominará jobs o trabajos a los procesos pertenecientes a un usuario concreto y, desde su punto de vista, habrá un trabajo en modo normal, "primer plano" o foreground, y podrá haber uno o más ejecutándose en paralelo en "segundo plano" o background. Para lanzar un proceso en background se le invocará seguido del carácter &, por ejemplo ls -l > dir &. Si un proceso en background genera salida, ésta aparece en el terminal por lo que si se desea evitarlo conviene redireccionarla. Si el trabajo es interactivo, es decir, requiere también entradas en tiempo de ejecución, cuando éstas se produzcan quedará congelado o en stop. Las shells de la familia csh así como la bash (la shell por defecto en Linux) permiten suspender trabajos temporalmente pulsando CTRL+Z.

Para el control de procesos, UNIX proporciona comandos como:

ps: Muestra los procesos actuales.
jobs: Muestra la lista de trabajos del usuario.

Es interesante destacar que existe una numeración única para cada proceso ejecutado por el sistema que es facilitada por ps y una numeración particular de los trabajos de cada usuario facilitada por jobs. Así, no es

```
 _      r w x     r w x     r w x
 |      \___/     \___/     \___/
TIPO    OWER      GRUPO     RESTO

• Tipo   s:  socket             c:  dispositivo de caracter
         l:  link o enlace      b   dispositivo de bloque
         d:  directorio

• Bit    1   (r)                Permiso lectura
• Bit    2   (w)                Permiso escritura
• Bit    3   (x)                Permiso ejecución
```

FIGURA 2: Estructura de las cadenas de protección de ficheros.

lo mismo el proceso 1 ó init, padre de todos los procesos, que el trabajo 1 de cada usuario, diferencia importante a la hora de, por ejemplo, aniquilar un proceso. Así, cada trabajo esta asociado a su jobid (job identifier) o número de trabajo propio para cada usuario, y a su pid (process identifier) o número de proceso común a todo el sistema, y cualquiera de ellos será valido para referirse al mismo. La referencia a un pid se realiza simplemente a través del propio número que constituye el pid, mientras que un jobid se distingue por ir precedido del signo %. Así, una posible salida de jobs sería:

(1) + Stopped vi factor.l
(2) - Stopped mail echeva
(3) Running lisp

y posibles jobids para estos trabajos serían:

%n, donde n=1,2,3;
%nombre, donde nombre=vi,mail,lisp;
%prefijo, donde prefijo=subcadena del nombre que hiciera distinguible al trabajo;
%+ , para referirse al último trabajo suspendido;
%- para referirse al anterior.

Otros comandos para control de trabajos son:

fg jobid: Trae al foreground un trabajo en background o Stopped.

bg: Pasa un proceso stopped (por ejemplo, por haber pulsado CTRL+Z) a background.

kill: Mata un trabajo o proceso. Si éste se resiste a morir, se le debe enviar la señal "asesina" -9: kill -9 jobid o kill -9 pid. Por ello, es importante distinguir que si el root ejecutara kill -9 %1 (jobid), produciría la muerte de su primer trabajo, pero si, en cambio, ejecuta kill -9 1 (pid), provocaría la muerte inmediata del proceso init y con él la caída total del sistema de forma desordenada con consecuencias, al menos, "no buenas". Si fuera cualquier otro usuario quien lo intentara, el propio sistema rechazaría la orden al no provenir directamente de root. Del mismo modo, si root ejecuta rm -r * desde el directorio raíz, acabará con el sistema de ficheros, mientras que si lo hace otro usuario, sólo borrará aquellos permisos para los que tenga permiso de escritura (generalmente, los suyos). Este es uno de los motivos para no actuar como root más tiempo del estrictamente necesario. Por supuesto, para no entrar al sistema como usuario habrá que contar con otra cuenta. La mayoría de las distribuciones de Linux, cuentan con una utilidad llamada adduser cuyo manejo se muestra en ADDUSER.TXT y que sirve precisamente para crear otras cuentas. En ella se hace referencia a un gid (group identifier) y a un uid (user identifier) que, por el momento será mejor dejar elegir al propio programa.

IDENTIFICACION DE OTROS USUARIOS

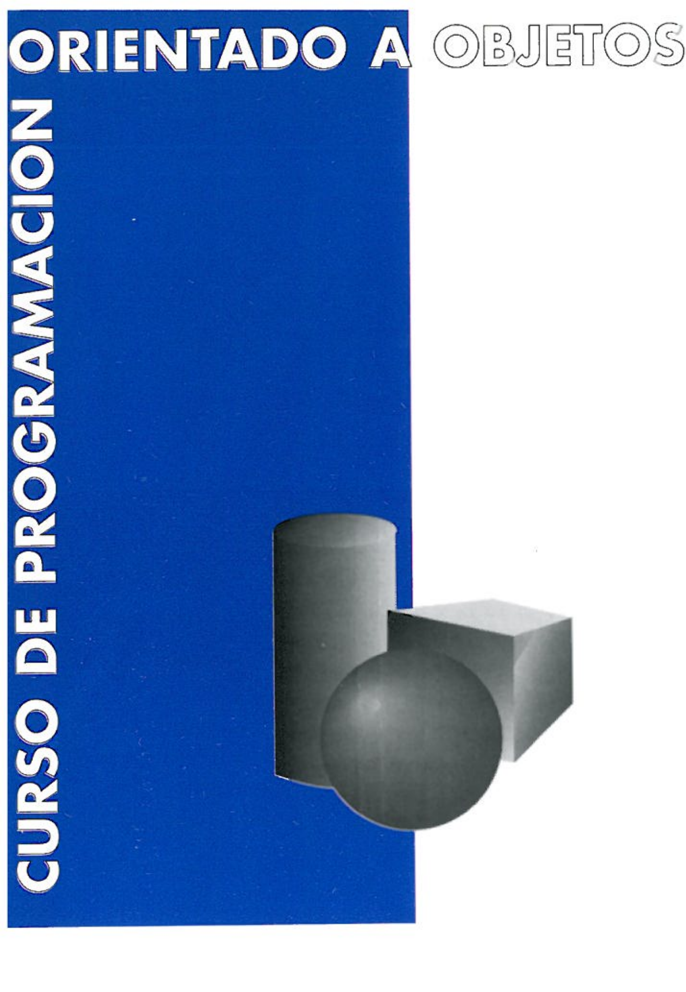
Existen dos comandos análogos con los que se puede identificar a otros usuarios conectados a la vez en la misma máquina: who y w, cuyas salidas pueden verse en SESION3.TXT. Esta información es actualizada dinámicamente por el sistema y se encuentra en el fichero /etc/utmp. Como puede apreciarse, la información de w es más completa y presenta los siguientes campos:

- user, con los login o nombres de usuario, que aparecerán repetidos si el usuario tiene abierta más de una sesión (por ejemplo, en linux, pulsando ALT junto con las teclas de función se puede saltar de una sesión a otra a través de terminales virtuales);
- tty, que muestra el terminal a que se encuentra asociada la sesión;
- login, que indica la hora de inicio de la misma;
- idle, que muestra el tiempo en minutos que ha transcurrido desde que el usuario pulsó por última vez una tecla;
- JCPU o tiempo de CPU utilizado por todos los procesos controlados por el terminal;
- PCPU o tiempo de CPU usado por el proceso activo,
- what o proceso activo de la sesión.

Para obtener información sobre un usuario, se encuentre éste o no conectado en ese momento a la máquina, se dispone del comando finger [name], que presenta información sobre todos los usuarios de login name o con la cadena name en su nombre real. En caso de teclearse sin argumentos, dará una información similar a who de todos los usuarios conectados a la máquina. La salida de este comando suele presentar un campo Project, que muestra la primera línea del fichero .project (en caso de que exista) del home directory del usuario por el que se pregunta y un campo Plan que muestra el fichero .plan completo.

EL PROXIMO CAPITULO

En los próximos capítulos se presentarán los mecanismos de comunicación entre máquinas y usuarios y se profundizará más para llegar al dominio de utilidades y shells. ■



STREAMS

Ignacio Cea

No cabe duda de que usted, en su época de programación bajo C, habrá empleado instrucciones como fprintf, fputs, fopen, fwrite o fclose para realizar las tareas de entrada/salida de información. Sin embargo, en un lenguaje como C++, en el que los nuevos tipos de datos pueden llegar a tener un comportamiento totalmente equivalente al de los datos estándar, tal forma de manejo se vuelve pobre y obsoleta.

Necesitamos caminos para poder manejar nuestros objetos de una forma análoga a como lo haríamos con un char o un int. Requerimos, en definitiva, una entrada/salida que pueda evolucionar y adaptarse a la existencia de nuevas clases.

El manejo de streams puede ocupar libros enteros. Este artículo no pretende agobiarle con tanta información sino tan sólo darle unas guías para mejor comprensión de unos futuros capítulos sobre el tema.

¿QUE ES UN STREAM?

Un stream no es, ni más ni menos, que un objeto, el cual hace de puente entre el resto de objetos de su aplicación y el dispositivo en el que piensan ser almacenados. Podemos compararlo con una cinta transportadora en la que se podemos tanto recoger objetos que vienen de arriba (un teclado) como introducirlos para que sean consumidos más abajo (una impresora).

Al hecho de introducir objetos dentro de un stream se le denomina inserción, mientras que extracción es el acto de sacar objetos del canal.

DOS STREAMS ESTANDAR

Dentro del fichero header fstream.h, se definen dos objetos streams, *cout* y *cin,* que comunican, respectivamente, la pantalla y el teclado con nuestras aplicaciones. Seguro que recuerda que durante las anteriores entregas hemos estado empleando ambos como si de nuevas órdenes del C++ se tratara. Es hora de que sepa que tanto uno como el otro no son comandos sino nombres de objetos de una clase.

La clase a la que pertenece el objeto cout tiene redefinido en su interior el comportamiento del operador << para poder realizar las operaciones de extracción, mientras que dentro de la clase al que pertenece cin el sobrecargado es >> para realizar las de inserción.

Tanto uno como otro operador están sobrecargados para poder manipular, tan sólo, los datos estándar del

En esta nueva entrega van a presentarse los streams, la nueva forma en la que el C++ entiende las operaciones de entrada/salida. Su potencia es soberbia, por lo que es conveniente que no se pierda el mínimo detalle y practique mucho con su máquina.

C++. Usted puede, además, sobrecargarlos para que también manipulen los nuevos tipos de datos que fabrica; es decir, sus propias clases.

ESQUEMA DE HERENCIA I/O

Las clases que dentro del C++ se ocupan de la entrada/salida son innumerables (del orden de 15). Todas derivan de una global denominada ios, como puede observar en la figura 1. Los objetos estándar cout y cin son instancias de las clases *ostream_withassign* y *ostream_withassign* respectivamente, que se encuentran en la parte más baja del esquema de herencia. No es tema de esta entrega el detallar cuáles son las responsabilidades de todas y cada una de esas clases, aunque sí el enseñarle cómo adaptarlas a sus propias necesidades.

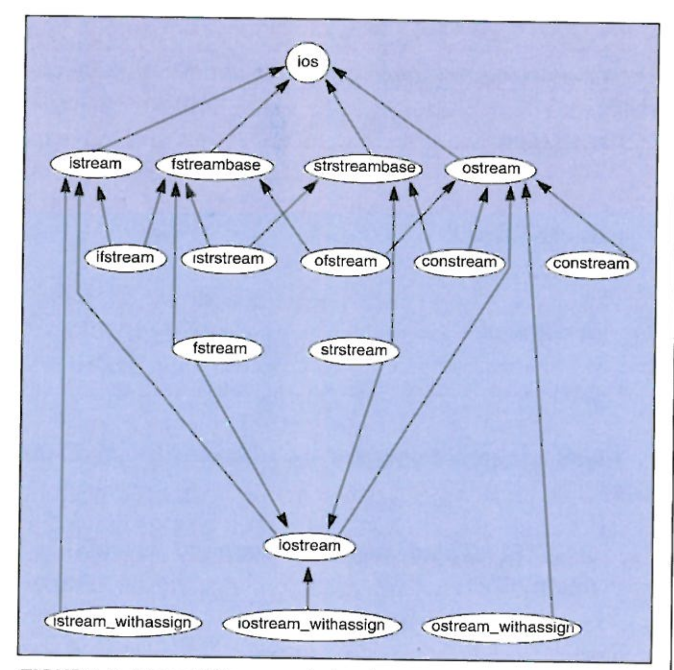


FIGURA 1. Clase IOS y sus derivadas.

EL OPERADOR DE INSERCION

Como se ha visto anteriormente, el operador << recibe el nombre de operador de inserción cuando se habla de streams. Esa funcionalidad viene derivada de la sobrecarga que del operador << se hace dentro de la clase ostream. Esa sobrecarga se hace a través de funciones miembro de la clase (ver entrega anterior), en la que el parámetro recibido es el tipo de dato a manipular y el valor de retorno una referencia al mismo objeto de la clase que ha realizado la inserción. Esto tiene el único fin de poder realizar operaciones de inserción en cadena. Cualquier canal que creemos miembro bien de esta clase, bien de las derivadas, podrá emplear ese operador para realizar la salida de datos. cout, sin duda, es un ejemplo de este caso:

```
....
operator FAR &ostream << (char);
operatir FAR &ostream << (int);
...
```

ADAPTAR EL OPERADOR DE INSERCION

Para permitir que sus aplicaciones también realizaran operaciones con los objetos de sus nuevas clases no tendría más que sobrecargar una vez más el operador <<. Para ello habría que añadir, dentro del fichero fstream del sistema y tras las anteriores órdenes, la siguiente línea de código:

```
...
operator FAR &ostream << (MyClass);
...
```

Imagine, sin embargo, el trastorno que la modificación de un fichero del sistema puede llegar a acarrear. Por tanto, para permitir que los objetos de una clase cualquiera de sus aplicaciones puedan ser insertados y extraídos dentro de streams como si de cualquier otro tipo básico se tratara tendremos que emplear un método friend (ver entrega anterior) dentro de la clase en cuestión. Así:

```
...
friend ostream &operator << (ostream
&,MyClass &);
...
```

Observe que el método friend, además de retornar la referencia al stream de salida empleado, también lo recibe como parámetro. Recuerde siempre que si sobrecargamos un operador desde dentro de la clase que lo emplea (en este caso sería ostream), tal referencia está por defecto, mientras que si la sobrecarga se hace desde la clase empleada (en este caso MyClass) es necesario tanto emplear un método friend como pasar de alguna manera la referencia a la clase fuente (ostream).

¿QUE DEBE HACER EL NUEVO OPERADOR INSERCION?

Cualquier clase de cualquier aplicación que usted desarrolle tendrá atributos en su interior. Dichos elementos serán, o bien variables de un tipo estándar del siste-

FIGURA 2

Manipulador	acción.
dec	coloca a 1 el flag de conversión a decimal.
hex	iden para hexodecimal.
oct	iden para octal.
Ws	extrae de la vigente valida los caracteres blanco.
endl	[SPC] [TAB] etc...
ends	Inserta un caracter '/0' en el stream.
jlush	Vacía el stream.
setbase (intn)	Coloca el formato de conversión a base n (0, 8, 10 ó 16). O significa el valor por defecto: Base 10.
Setw (intn)	Coloca la anchura del siguiente campo a n caracteres.

Manipuladores genéricos más importantes.

ma (int, char, char *), o bien objetos miembro de otras clases (esta última forma de construcción se denomina agregación).

Insertar un objeto en un canal suele implicar la inserción de todos y cada uno de sus atributos miembro. Ni que decir tiene que para poder hacer la inserción de un atributo objeto también la clase a la que aquél pertenece ha de tener sobrecargado el operador <<.

Evidentemente, esto es una regla muy general y cada programador puede limitarse, básicamente, a hacer lo que quiera dentro de esa sobrecarga.

Observe los ejemplos 1 y 2 para ver ejemplos sobre estas cuestiones.

EL OPERADOR DE EXTRACCION

El operador de extracción (>>) se comporta de forma completamente análoga a la expuesta para el operador << en párrafos anteriores. La única diferencia estriba en que la clase stream manipulada es istream en lugar de ostream.

Mire detenidamente los ejemplos 3 y 4 para una mejor compresión.

FIGURA 3

Manipulador	Función conio equivalente	acción
CL reol	CL reol	Limpia hasta el final de la línea en una ventana texto.
del line	del line	Borra la línea entera en una ventana texto.
high video	highvideo	Coloca los caracteres en high-intensity.
insline	insline	Inserta una línea en blanco dentro de una ventana texto.
lowvideo	lowvideo	Coloca los caracteres en low-intensity.
normvideo	normvideo	" " " " " "
Setattr (intn)	texattr	Coloca atributos de pantalla texto.
Setbk (intn)	texcolor	Nuevo color para los caracteres.
Setclr (intn)	textcolor	" " " " " "
Setcrsxtype (intn)	Setcursor type	Selecciona nueva apariencia para el cursor.
Setxy (int, int)	gotoxy	Posiciona el cursor dentro de una ventana de texto.

Manipuladores para streams de pantalla.

UN DETALLE IMPORTANTE

No olvide que los objetos desde los que se invocan los operadores << y >> son siempre miembros, respectivamente, de las clases ostream e istream. cout y cin son, tan sólo, dos de ellos que se encargan del manejo de pantalla y teclado.

Sin embargo, usted podría crear sus propios canales para tratar con otros periféricos como, por ejemplo, la impresora o el disco duro. Para esos streams puede emplear, igualmente, los operadores de inserción y extracción.

No haga, por tanto, sus sobrecargas excesivamente dependientes de cout o cin, pues los podría llegar a hacer inservibles para tratar con cualquier otro tipo de canales.

FIGURA 4

Bit	Acción
105:: app	Añade datos al final fichero.
105:: ate	Salta al final del fichero una vez que es abierto.
105:: in	Abre para entrada (por defecto de ifotream).
105:: out	Abre para salida (por defecto en ofstream).
105:: binary	Abre el fichero en modo binario.
105:: trunc	Descarta el contenido de un fichero si quél existe (por defecto si colocamos 105:: out y ni los:: ate ni 105:: app son especificados).
105:: no create	Si al abrir el fichero aquél no existe se genera error.
105:: noreplace	Si el fichero existe, open falla a menos que ate o appe sean colocados.

Formas de creación de fstreams.

Por ejemplo:

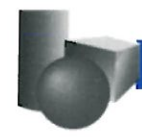
```
...
class MyClass {
 private :
  ...
  int numero;
  ...
  public :
  ...
 friend ostream &operator << (ostream &O,MyClass
&M)
  {
   O << "El número vale: " << M.numero << endl;
   return (O);
  }
  ...
  };
```

es excesivamente dependiente de cout. Parece una clase hecha para tratar sólo con él. En su lugar se debería emplear una fórmula como:

```
...
friend ostream &operator (ostream &O,MyClass &M)
{
 O << M.numero;
 return (O);
}
...
```

FORMATEANDO LAS OPERACIONES

Incluso la vetusta función printf podía formatear la salida de datos, cosa que, hasta el momento, parecen no permitirnos los streams. Sin embargo, dentro de ellos también existe la posibilidad de formatear tanto la inserción como la extracción de datos estándar (y sólo de datos estándar).



Eso se consigue a través de lo que en C++ se denominan manipuladores (manipulators). Los manipuladores siempre se colocan entre la inserción o extracción de dos datos de tipo estandar (int, char, etc...) y sólo afectan a la inserción o extracción que va a continuación de ellos.

Los manipuladores pueden llegar a alterar por completo el formato en el que será insertado o extraido el siguiente dato. Por ejemplo, la siguiente línea de código:

```
cout << setw (6) << setfill ('*') << 6 << endl;
```

visualizará:

```
*****6 <RETURN>
```

En la figura 2 podrá encontrar una lista completa y detallada de todos los manipuladores definidos dentro del C++. Además de los ahí expuestos existen otros tantos que afectan sólo a los streams de manejo de pantalla (como cout) y que vienen en cierta medida a reemplazar a las vetustas funciones C de control de pantalla definidas en conio. En la figura 3 encontrará referencias de todos ellos.

STREAMS DE USUARIO

Es hora de ahondar aún más en el complejo mundo de los streams y enseñarle a definir sus propios canales de comunicación. Empecemos por los más "sencillos". Por ejemplo, aquellos que se ocupan de manejar ficheros en disco.

Lo primero que tendrá que hacer es crear un stream. Para ello lo único que ha de especificar es el nombre del fichero con el que quiere comunicarse y el tipo de enlace que desea efectuar con él. Ambos parámetros son similares a los que recibía la antigua función del C fopen. En la figura 4 podrá encontrar una lista de todos los distintos modos de enlace definidos en el sistema de streams de C++.

Esas formas de comunicación pueden encadenarse mediante el operador | (or) de manera análoga a como se hacía en fopen. Ese parámetro, el segundo en la creación del stream, es opcional, ya que cualquier tipo de stream dará siempre alguno por defecto. Así por ejemplo, un ostream se abrirá como canal de salida, mientras que un istream lo hará como de entrada.

Algunos de esos parámetros son, entre sí, opuestos, lo cual implica que el encender uno implicará el apagado de otros y viceversa. Las posibilidades son tantas que lo único que se puede recomendar es la propia experimentación.

Observe los ejemplos 5, 6 y 7 para comprender mejor el manejo de streams.

LOS METODOS DE STREAM

Dentro de cualquier clase stream hay definidos infinidad de métodos que se ocupan de manipular la información que circula por el canal. De entre todos ellos destacan close y open, que permiten abrir y cerrar momentáneamente el canal.

¡Cuidado!: cerrar el canal no implica cerrar el fichero. El fichero sólo se cerrará cuando finalice el ámbito de validez del objeto stream que lo maneja (cuando se destruya el objeto). Dentro del los ejemplos 5, 6 y 7 se manejan estos métodos.

FIGURA 5

Bit	Acción
105:: beg	Referencia desde el comienzo del fichero.
105:: end	" " " el final " " .
105:: cor	" " " la posición actual del cursor en el fichero.

Puntos de referencia en punteros fstream binarios.

Observe que para realizar tanto la salida como entrada de información se emplean los mismos operadores que con cout y cin... ¿Se da cuenta de la potencia que ello conlleva?

UN PROBLEMA

Parece, y es cierto, que el manejo de streams aquí presentado funciona sólo como archivos secuenciales de información. Pero, ¿qué sucede si queremos grabar información en un fichero binario?

LOS FICHEROS BINARIOS

Para crear un stream que trate los ficheros como binarios basta indicárselo en el segundo parámetro del constructor (ios::binary).

Un stream binario lleva siempre asociados dos punteros o cursores que se emplean para recorrer el fichero de un extremo a otro. Ambos son independientes entre sí. El primero servirá y se verá afectando por las operaciones de escritura, mientras que el segundo lo hará con las de lectura.

En el interior de cada stream hay métodos para manipularlos. Evidentemente, dentro de los ostreams sólo habrá métodos para manejar el puntero de escritura mientras que en los istreams sólo existirán los que manipulan el de lectura. En los fstreams, sin embargo, se incluyen ambos.

El puntero de lectura se posiciona mediante el método seekg, mientras que el de escritura lo hace con seekp. Tanto uno como otro reciben dos parámetros: el primero de ellos es un long que indica el número de byte del fichero sobre el que han de colocarse, mientras que el segundo señala la posición del fichero que se toma como origen de la anterior referencia. Dentro de la figura 5 se detallan los posibles puntos de referencia.

En contraposicion a estos dos métodos, tellg y tellp permiten consultar la posición de los cursores de lectu-

ra y escritura, respectivamente, dentro del fichero que estamos manipulando.

Las operaciones de lectura o escritura binaria no pueden hacerse a través de los operadores << ó >>, ya que tanto uno como otro operan sólo sobre ficheros formateados, que es el caso de los vistos hasta ahora (el 13 no seria un byte 13 sino un retorno de carro). En su lugar hacer aparición write y read, en todo análogas a las del C.

Tanto write como read reciben como parámetro un puntero char o unsigned char (recuerde lo estricto que es el C++ para los tipos), y el número de bytes que se quieren escribir o leer. No olvide que write sólo estará vigente en aquellos objetos streams que permitan la escritura (istream no), y que read sólo aparecerá en los de lectura (ostream no).

Observe el ejemplo 8 del dsico, para la manipulación de streams binarios.

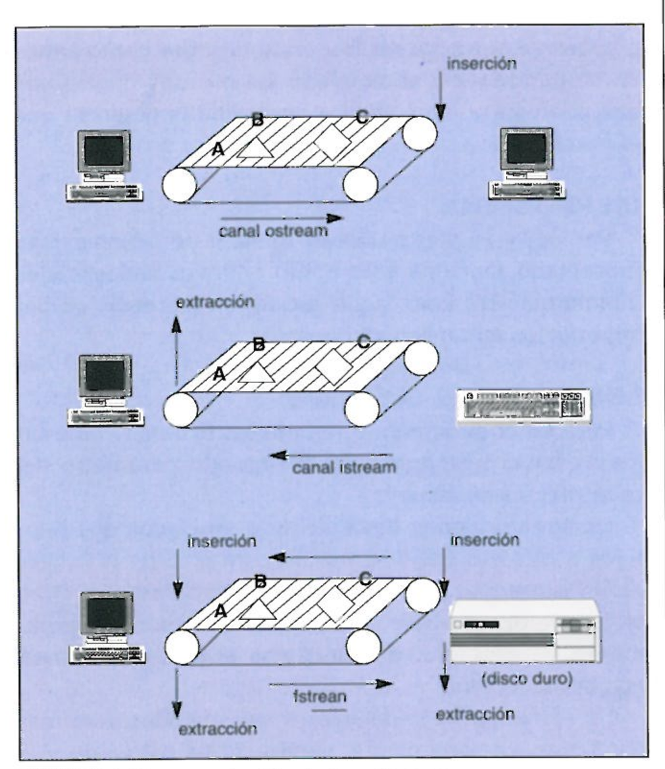


FIGURA 6. Representación de los streams.

EL MANEJO DE ERRORES

Seguro que se ha percatado de que en ningún momento hemos hablado de la información que retornan todos y cada uno de los métodos a los que nos hemos referido. La mayor parte de ellos devuelven una referencia al objeto stream que los ha empleado:

```
ostream &write (unsigned char *,long);
```

Todas las clases stream tienen a su vez sobrecargado el operador ! para comprobar si la última operación efectuada sobre ellos ha generado un error. Retorna 0 si el stream está correcto y 1 si no. La combinación del valor de retorno con el operador ! puede permitirnos controlar los errores de sus operaciones:

```
if (!fichero.write ((unsigned char *) MyObjeto
        sizeof (MyObjeto))) {...}
```

Una vez que hemos comprobado que se ha producido un error podemos, a través de ciertos métodos definidos dentro de la clase ios (de la que derivan todos los streams), determinar de qué error exactamente se trata.

Dentro de la clase ios hay una serie de bits que se colocan a 1 o a 0 según el error. Con esos métodos podemos consultar individualmente o en grupo cada uno de esos bits. En la figura 6 puede encontrar una referencia de todos esos métodos y bits. Aquellos métodos que no devuelven una referencia al objeto que los emplea al invocarlos es porque, o bien se trata de un constructor, o bien devuelven otro tipo de información útil para el programador. Aún así, el error puede ser consultado antes de hacer cualquier otra operación:

```
long pos = fichero.tellg ();
if (!fichero) cout << "No puedo ir ahí" << endl;
```

INCLUDES

Para poder manejar cualquiera de los tipos de stream expuestos en este artículo ha de incluir previamente el fichero fstream (formated stream). Y si además va a usar manipuladores, debe añadir el iomanip.h.

LINEAS DE INVESTIGACION

Como habrá podido observar, en los streams hay un vasto campo de investigación. Lo que aquí se le ha expuesto es tan sólo una introducción al mismo, que es más que suficiente para que haga sus pinitos en la materia. En próximo capítulos, al entrar en ejemplos prácticos, ahondaremos en algunos puntos.

COMO COMPILAR LOS EJEMPLOS

Todos los ejemplos han sido desarrollados empleando el Borland C++ 3.1. Sin embargo, los streams es algo tan estándar que su compilación en cualquier otro sistema no puede ofrecer ninguna dificultad. Es de notar que no hay diferencias entre el manejo de streams en un sistema UNIX y en una máquina DOS, al contrario de lo que sucedía con los ficheros del C.

Para ejecutar un ejemplo basta cargar el fichero que contiene al mismo y pulsar Run. Le recomendamos que si su sistema dispone de Debugger ejecute todas las aplicaciones paso a paso.

PROXIMA ENTREGA

En la próxima entrega nos ocuparemos de los template. Algo incorporado dentro de la última revisión ANSI del C++ que permite la definición de plantillas de clases para reusar aún más si cabe el código generado por este lenguaje. ■

CORREO DEL LECTOR

En esta sección, los lectores de SOLO PROGRAMADORES tienen la oportunidad de hallar respuesta a casi todos sus problemas relaccionados con la programación de ordenadores o referentes a los artículos publicados en esta revista.

P *En muchos programas en C++ he observado algunas rutinas dentro de varias clases con el siguiente formato:*

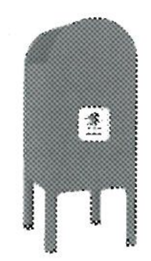
```
NombreClase (const NombreClase &x)
{
...
};
```

¿Que utilidad tiene el crearla con el mismo nombre de la clase y coger en la entrada una referencia constante a un objeto de su mismo tipo? Por otro lado, ¿pueden tener parámetros los destructores?

Carlos Herrera (Barcelona)

R En C++, a las rutinas se les denomina métodos, ya que son inherentes a cada clase. Los que ha visto se llaman constructores copia y se encargan de copiar los datos de un objeto a otro. El constructor copia siempre está presente en cada clase, pero como el compilador crea uno por defecto que suele funcionar correctamente para los fines deseados, no suele ser necesario definirlo, aunque ciertos desarrolladores siempre lo definen para "tener el control total del programa". El constructor copia puede tener más parámetros de entrada, pero deben ser todos por defecto. Pongamos un ejemplo:

```
class persona
{
private:
  int edad, estatura, peso;
  public:
  // Constructor por defecto
  persona (int a=0, int b=0, int c=0)
  {edad=a; estatura=b; peso=c;}
  // Constructor copia
  persona (const persona &per)
  {
   edad=per.edad;
   estatura=per.estatura;
   peso=per.peso;
  }
}
```



El parámetro del constructor copia se ha definido como constante; esto evita la construcción temporal del parámetro en tiempo de ejecución. Si se crean objetos del tipo persona ocurre lo siguiente:

```
persona yo;
persona tu (29, 180, 80);
persona el (tu);
yo = tu;
```

En el primer y segundo caso se llama al constructor por defecto (el que se ha definido), pero en el tercer caso, como se está creando un objeto a raíz de los datos de otro, se ejecutará el constructor copia después de crear la variable, y pasará a tener los mismos valores que el objeto de entrada. ¿Que ocurre en el cuarto ejemplo? Pues como la variable ya está previamente creada, se llamará al operador de asignación en vez de al constructor copia (este método también suele venir implementado automáticamente por el compilador). La diferencia entre ambos es que el constructor copia crea un nuevo objeto, y el operador de asignación funciona con objetos ya creados, aunque ambos dupliquen los valores internos de las clases.

P *Cuando quiero poner una pausa en mis programas, esperando la pulsación de una tecla cualquiera, a veces la espera no se realiza, pues el buffer de teclado me envía una pulsación antigua o un código extraño. ¿Cómo puedo evitar esto? Gracias y adelante en nuestra/su revista.*

Antonio Muñoz (Madrid)

R Seguro que no limpia el buffer de teclado antes de leer cualquier cosa. El método es muy sencillo y obligatorio de cara a eliminar algunos scan codes dobles enviados por las teclas especiales del PC. Sólo debe leer tantas veces como el ordenador "se lo pida":

```
void limpteclado(void)
{
 while (kbhit()) getch();
}
```

P *Estoy empezando a programar en ensamblador (antes desarrollaba en C), y algunas veces me pierdo, casi siempre por falta de información. Quiero crear y manejar en ensamblador una interrupción que se genere cada cierto tiempo, por ejemplo para ir actualizando un contador más o menos exacto y rápido.*

Adolfo Conejero-Martín (Burgos)

R Su problema es habitual en la gente que empieza en ensamblador; no está nada hecho y no suele haber librerías estándar con cientos de utilidades para todo.

Primero se reprograma el timer del PC a la velocidad con la que se desee ejecutar la rutina en interrupción (en este ejemplo se elige 70 veces por segundo). Para calcular el número de interrupciones por segundo se debe dividir la velocidad a la cual corre el timer: 1234DDh o 1193181 en decimal (en Hz) entre el número de interrupciones por segundo que se deseen: 1193181 / 70. Luego se coloca la rutina a ejecutar en el vector de interrupción correspondiente (el 8 para el timer) y se ejecutará el número de veces deseado realizando la tarea encomendada. Si la frecuencia de interrupción es muy elevada, es preferible que la rutina sea lo más sencilla y rápida posible, pues si no la velocidad del sistema se vendrá abajo.

```
movcx,17045           ; int. por segundo
call settimer
movax,8               ; vector interrupcion
movcx,cs
movdx,offset cuenta
call write vector
;Entradas:
;cx:dx= dirección nueva rutina de interrupcion
;ax= numero de vector a modificar
write vector:
cli
push        ds
xor bx,bx
movds,bx
add ax,ax
add ax,ax
movbx,ax
mov[bx],dx
mov[bx+2],cx
pop ds
sti
ret
; Entrada: cx.-int por seg.
settimer:
cli
moval,00100100b
out 43h,al
moval,cl
out 40h,al
moval,ch
out 40h,al
sti
ret
;Interrupcion a colgar del timer
cuenta:         sti
push            ax
inc [flagcontador]
cli
moval,20h    ; fin interrupción
out [20h],al
pop ax
iret
```

P *En mis códigos tengo varias interrupciones activas y cuando algo falla, me vuelvo loco buscando la causa, pues las características de los depuradores en el manejo y captura de las interrupciones deja mucho que desear. Uso varios trucos para controlar en qué punto del código está mi programa, pero me ralentizan enormemente la velocidad, pues mis interrupciones tienen frecuencias muy altas. ¿Conocen algún depurador o sistema para visualizar la posición actual del IP o el tiempo que se gasta en una interrupción? Gracias y adelante con la revista.*

Borja Marciel (Madrid)

R Existen utilidades del tipo profilers que verifican el tiempo que la aplicación gasta en cada rutina y generan una estadística final para observar dónde se "aburrieron" los registros, pero también existen varios trucos para verlo visualmente en tiempo de ejecución. Uno de los predilectos es el siguiente, que se debe llamar al principio y al final de cada interrupción. Con él se observarán unas bandas de color más o menos estables que muestran el tiempo consumido por cada interrupción en forma de porciones de pantalla (el tiempo que el haz de electrones de la tarjeta de video tarda en "dibujar" una zona de pantalla), pero si la rutina de interrupción a comprobar es muy larga, la visualización no es muy exacta.

```
void poncolor0(char color)
{
inp(0x03da);             // Modo índice
outp(0x3c0, 0x11);       // Leer registro
outp(0x3c0,color);       // Activa el registro del color 0
inp(0x03da);             // Modo índice
outp(0x3c0,0x20);        // Paleta accesible de nuevo
}
```



P *¿Cómo puedo leer el estado del ratón desde mis programas en Pascal? ¿Es estándar este procedimiento? Gracias por todo.*

Antonio Orallo (Toledo)

R Sí, las llamadas al driver del ratón son cada vez más estándar y sólo variarán un poco según se trate de ratones de tres botones (tipo Genius) o de dos (tipo Microsoft). La interrupción que contiene las funciones de manejo del ratón es la 0x33 ($33 en Pascal) y existen funciones para realizar cualquier proceso, pero con conocer un par de ellas es suficiente.

Las principales llamadas al driver del ratón son:

Funciones	Número
Reseteo del Driver	0
Activar visualización del cursor	1
Desactivar visualización del cursor	2
Leer estado y posición del ratón	3
Recolocar el ratón	4
Definir rango horizontal (X) del cursor	7
Definir rango vertical (Y) del cursor	8
Definir estilo del modo gráfico del cursor	9
Definir estilo del modo texto del cursor	10
Definir sensibilidad del ratón	15

Ejemplo de llamada a una función del ratón en Pascal:

```
const MOUSEINTR = $33;
procedure EstadoRaton (var x, y : word; var boton1,
boton2 : boolean);
var regs : registers;
begin
        regs.ax := 3;
        Intr (MOUSEINTR, regs);
        x := regs.cx;
        y := regs.dx;
        boton1 := (regs.bx and 1) <> 0;
        boton2 := (regs.bx and 2) <> 0;
end;
```

Lo mismo, pero en C:

```
#define MOUSEINTR   0x33
void EstadoRaton (int *x, int *y, char *boton1, char
*boton2)
{
        union REGS regs;
        regs.x.ax = 3;
        int86 (MOUSEINTR, &regs, &regs);
        *x = regs.x.cx;
        *y = regs.x.dx;
        *boton1 = regs.x.bx & 1;
        *boton2 = regs.x.bx & 2;
}
```



BOLSA DE TRABAJO

BOLSA DE TRABAJO
BOLSA DE TRABAJO
BOLSA DE TRABAJO

En esta sección los lectores tienen la oportunidad de publicar gratuitamente sus demandas y ofertas de empleo. Para aparecer en esta página, envíen un fax o carta a la revista, haciendo referencia a "Bolsa de Trabajo, Sólo Programadores".

TU OPORTUNIDAD CON SOLO PROGRAMADORES

- **Jesús Vaquero.** Saubrigues (Francia). REF: 39.
  - Titulado en Electrónica e Instrumentación..
  - Sculptorm Clarion, ADA, AIX, VReam, WTK.
  - Inglés, Francés.
- **Juan Luis García.** Avilés. REF: 40
  - FPII. Técnico especialista en Informática de Gestión.
  - Hardware PC, Cobol, C, D Base.
  - Inglés.
- **Susana Toledano.** Madrid. REF: 41.
  - Diplomada en Informática. Esp Sist. Lógicos.
  - Cobol, Modula-2, SQL, C++, UNIX.
  - Inglés, Francés.
- **Cristina Salmerón.** Talavera de la Reina. REF: 42
  - Ingeniería T. de Informática de Sistemas.
  - UNIX, AIX, Informix, Ingres, C++, Ensamblador.
  - Inglés.
- **Luis Saucedo.** Chiclana de la Fra. REF: 43
  - FPII. Técnico especialista en Informática de Gestión.
  - Basic, Ensamblador.
  - Inglés, Francés.
- **Angel Perez.** Sevilla. REF: 44
  - Licenciatura de Informática..
  - Basic, C, Cobol, UNIX
  - Inglés.
- **Alberto Mañas.** Madrid. REF: 45
  - Diplomado en Informática.
  - Prolog, Lisp, SQL, Oracle, Informix.
  - Inglés.

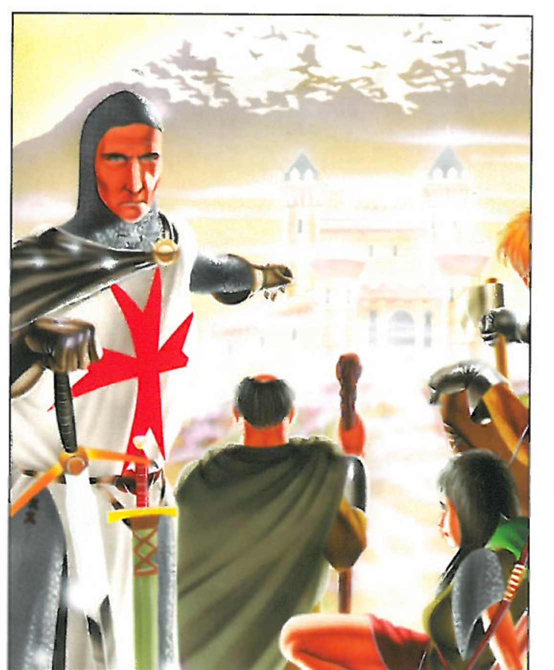
OUT FOR LUNCH

MEJOR CON ORDENADOR, PERO TAMBIÉN SIN ÉL

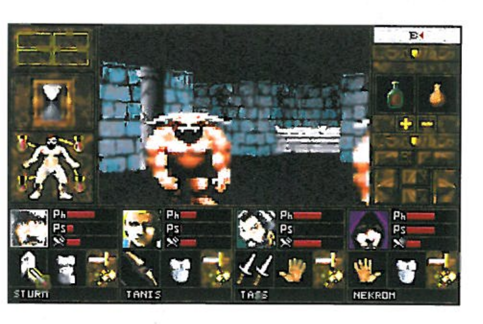
ROL CRUSADERS



Gráficos realizados por auténticos artistas.



Escenarios cuidados hasta el más mínimo detalle.



Movimientos rápidos y precisos en 3D.



Escenas digitalizadas animan el desarrollo del juego.




Gran cantidad de armas y objetos.



Enigmáticos personajes proporcionan valiosas indicaciones.

POR FIN TODO LO QUE DESEAS DE UN JUEGO DE ROL, REUNIDO EN UN MISMO PAQUETE ! MAS DE 7 MB DE INGENIO, AVENTURA Y ACCION TE HARAN PASAR RATOS INOLVIDABLES.

DISFRUTA DE LOS MEJORES GRAFICOS Y LOS SONIDOS MAS ESPECTACULARES CON TUS AMIGOS. ELIGE, UNO A UNO, LOS COMPONENTES DE TU CUADRILLA, Y PREPARATE A AFRONTAR LOS PELIGROS QUE SE TE AVECINAN, Y SI SE VA LA LUZ...

¡NO TE PREOCUPES! ROL CRUSADERS ES EL UNICO JUEGO DE ROL PARA ORDENADORES QUE INCLUYE ADEMAS UNA VERSION EN JUEGO DE MESA CON DADOS, LIBRO Y TABLERO.



CON LA GARANTÍA DE



¡Pídelo YA en tu quiosko!
(por sólo 2.995 pts)

Solicita ROL CRUSADERS enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 16 a 18:30

…o que me envíen **ROL CRUSADERS** al precio de 2995 ptas + 250 ptas. de gastos de envío.
…bre y apellidos..Domicilio..Población..................................
…incia...C.P ....Fecha de nacimiento...............................Profesión...

FORMA DE PAGO:
Talón a TOWER COMMUNICATIONS S.R.L. ☐ Contra reembolso (+ 250 Ptas. adicionales de gastos de envío)
Giro Postal nº...de fecha..........................
Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ Firma ,
AMERICAN EXPRESS nº ☐☐☐☐ ☐☐☐☐☐ ☐☐☐☐☐
Fecha de caducidad de la tarjeta.................Nombre del titular, si es distinto...

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS S.R.L.
C/ Marqués de Portugalete, 10
28027 Madrid.



MINISTERIO DE DEFENSA
DIRECCION GENERAL DE LA GUARDIA CIVIL
MINISTERIO DE CULTURA
MINISTERIO DE ECONOMIA Y HACIENDA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
GENERALITAT DE CATALUNIA. DEP. SANITAT
SERVEI CATALA DE LA SALUT
BOLETIN OFICIAL DEL ESTADO



BANCO DE SANTANDER
INSTITUTO NCNAL. DE LA SEG. SOCIAL
MINISTERIO PARA LAS ADMINISTRACIONES PUBLICAS
FONDO DE GARANTIA DE DEPOSITOS
BANCO SANTANDER PUERTO RICO
BANCO ATLANTICO
PRICE WATERHOUSE
TOSHIBA



TELEFONICA
SANTANDER NATIONAL BANK
CORPORACION FINANCIERA HISPAMER
ANALISTAS FINANCIEROS INTERNACIONALES
SEGUROS OCASO
EUROSEGUROS BBV
SEGUROS ATHENA
FYCSA (ALCATEL)



CONSTRUCCIONES AERONAUTICAS (CASA)
PATENTES TALGO
SINTEL
AVIACO
AEROPUERTO DE CANARIAS
TRANSMEDITERRANEA
PUERTOS DEL ESTADO
GEC ALSTHOM
L OREAL

FASA RENAULT
RED DE CONCESIONARIOS RENAULT
REPSOL EXPLORACION
REFINERIA DE GIBRALTAR
PETROGAL
ONDA CERO RADIO
TELEMADRID



TELECINCO (GESTEVISION-TELECINCO)
UNIVERSIDAD AUTONOMA DE MADRID
UNIVERSIDAD CARLOS III DE MADRID
INSTITUTO DE EMPRESA
TELEFONICA SISTEMAS*
S.P. EDITORES*
OLIVETTI*



IBM*
SIEMENS NIXDORF*
DIGITAL*
BULL ESPAÑA*
FUJITSU*
INVESTRONICA*
SOFTWARE DE DIAGNOSTICO*



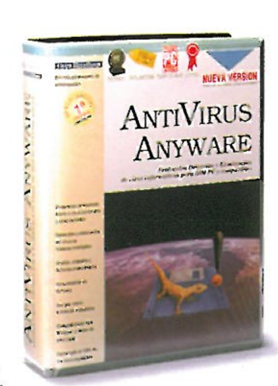
COMPUTER ASSOCIATES
G.P. INFORMATICA*
INFORMATICA EL CORTE INGLES*
ACTION*
GTI*
ALCATEL SISTEMAS DE INFORMACION*
INDAS



EL CORTE INGLES*
SOFTWARE DE ESPAÑA*
INFORMIX
ABBOTT CIENTIFICA
ALBILUX
ELIDA GIBBS
OXFORD UNIVERSITY PRESS

9 de cada 10 Ordenadores prefieren ANTIVIRUS ANYWARE.

El resultado está a la vista. O, ¿cree que tantas Empresas pueden estar equivocadas?. Tienen razones para no estarlo.



1. MAYOR PROTECCION.
Incorpora un Sistema de Protección con una ocupación mínima en RAM. Detecta el Virus antes de que pueda introducirse e impide el uso del fichero contaminado, avisándole a través de una ventana de alarma.

2. DETECCION INSTANTANEA.
De forma rápida y precisa, analiza cualquier unidad, directorio o fichero. Comprueba si su disco duro o disquetes contienen algún Virus. Chequea ficheros (incluso comprimidos), sector de arranque, tabla de particiones y unidades de red.

3. ELIMINACION DEFINITIVA.
Elimina los Virus àllí donde se encuentren, sin dañar los ficheros.

4. SERVICIO DE ACTUALIZACION PERMANENTE.
Usted puede recibir las nuevas versiones cómodamente en su domicilio o capturarlas vía modem.

ANYWARE pone a su disposición el CLUB DE USUARIOS HELP VIRUS con una HOT LINE exclusiva para consultas, noticias, BBS, etc.

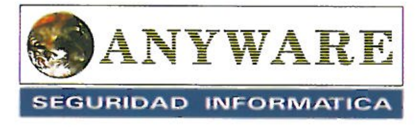








PREMIO PC MAGAZINE MEJOR UTILIDAD 1991 · RECOMENDACION PC WORLD MEJOR PRODUCTO ANTIVIRUS · RECOMENDACION PC MAGAZINE MEJOR PRODUCTO ANTIVIRUS · CATACOM 93 MEJOR ANTIVIRUS · PREMIO PC MAGAZINE MEJOR UTILIDAD 1993



Orense, 36 3º. 28020 MADRID. España.
Tel.: (91) 556 92 15. Fax.: (91) 556 14 04.

EMPRESAS E INSTITUCIONES CON LICENCIA CORPORATIVA DE ANTIVIRUS ANYWARE
(*)EMPRESAS DISTRIBUIDORAS DE ANTIVIRUS ANYWARE.